LA RECONNAISSANCE DE CARACTÈRES MANUSCRITS PAR RÉSEAU NEURONAL À FONCTIONS RADIALES DE BASE MUNIES D'ÉTATS

par

Pierre-Louis Constantin

mémoire présenté au Département de mathématiques et d'informatique en vue de l'obtention du grade de maître en informatique (M.In.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, août 1997

Le 10 novembre 1997 , le jury suivant a accepté ce mémoire dans sa version finale.

MEMBRES	*SIGNATURES*
M. Abdelhamid Benchakroun Département de math.-info.	_____
M. Djemel Ziou Département de math.-info.	_____
M. Shengrui Wang Département de math.-info.	_____

SOMMAIRE

Le réseau neuronal à fonctions radiales de base est un type de réseau récent dont le domaine d'application est vaste. Nous avons étudié les performances de ce réseau lors d'une utilisation pour la reconnaissance de caractères manuscrits. Nous avons obtenu un taux de reconnaissance de 89% des caractères présentés avec une forme du réseau simple et à architecture fixée a priori.

Nous avons ensuite énoncé les variantes possibles sur ce réseau neuronal et mis en lumière celles qui sont les plus utiles, particulièrement pour la reconnaissance de caractères manuscrits.

A partir de ces modifications et des résultats obtenus, nous avons développé un algorithme à architecture non-fixée a priori et optimisé pour la tâche de la reconnaissance de caractères manuscrits.

Les réponses de cet algorithme sont supérieures à celles de l'algorithme original en échange de plus de temps d'apprentissage. Il fait une recherche de solution d'une façon aussi bonne ou meilleure que certains algorithmes plus traditionnels mais a l'avantage d'offrir une solution localement optimale et de diminuer les tâches de l'architecte du réseau.

REMERCIEMENTS

Je voudrais tout d'abord exprimer ma gratitude au professeur S. Wang pour ses conseils judicieux et sa patience.

J'aimerais aussi remercier ma famille et mes amis pour m'avoir soutenu et encouragé en tout temps.

Je voudrais finalement féliciter les membres du groupe de recherche Neuro-Vision et particulièrement le professeur D. Ziou, pour avoir créé ensemble un groupe dynamique et enrichissant.

Table des matières

LISTE DES ABRÉVIATIONS

FRBE: réseau à fonctions radiales de base munies d'états.

k-NN: k-nearest neighbors, la règle des k plus proches voisins.

KRE: Kernel regression estimators, estimateurs de noyaux de régression.

LKMA: Local k-means algorithm, algorithme des k moyennes locales.

LVQ: Learning vector quantization.

MSE: Mean square error, moyenne de l'erreur au carré.

NIST: National Institute for Standards and Technology.

RBF: réseau à fonctions radiales de base, de l'anglais Radial Basis Function Neural Network.

SNNS: Stuttgart neural network simulator.

Liste des tableaux

Liste des figures

Introduction

Les réseaux neuronaux et la reconnaissance de caractères

Le réseau neuronal nommé RBF [1], mis en évidence par Broomhead et Lowe [10] en 1988, est rapidement devenu un des plus populaires grâce à sa simplicité et à ses performances. L'opération d'un réseau RBF est très simple car ses structures sont efficaces et faciles à comprendre. Ce réseau requiert très peu de ressources matérielles et est très rapide.

Le réseau RBF offre beaucoup d'avantages comparativement à d'autres réseaux plus traditionnels. Par exemple, il est plus petit et plus performant que le perceptron multicouche à rétropropagation du gradient. Une autre comparaison est que le réseau nécessite moins de mémoire que des modèles probabilistes multimodaux et ses capacités de généralisation sont aussi bonnes. Il est uti-

1. Radial Basis Functions: Fonctions radiales de base

lisé pour des applications aux problèmes d'apprentissage supervisé tels que la régression, la classification et la prédiction. Ses performances lui donnent un rôle important dans des applications en temps réel [5].

Le RBF est un modèle de réseau à trois couches (Entrée, Cachée, Sortie) à propagation avant avec fonctions d'activation de la couche cachée localisées. La méthode d'apprentissage du réseau RBF est de forme hybride semi-supervisée. La couche de sortie doit être entraînée de façon supervisée (l'utilisateur signifie quel résultat est attendu), mais la couche cachée est auto-organisée.

Le réseau RBF utilise des données locales pour faire l'apprentissage et le rappel. On utilise le terme *données locales* car, pour chaque valeur en entrée, seulement une petite région de l'espace du domaine du problème, plutôt que tout l'espace, entre dans le calcul. Dans le cas de RBF, plus une donnée entrée est éloignée des données déjà connues et bien classées, moins cette entrée a de l'influence sur le résultat du réseau. Ainsi on ne s'intéresse qu'aux données proches du vecteur d'entrée, les données locales.

Le système nerveux possède plusieurs types de neurones qui illustrent le principe du stimuli local. On peut penser aux stéréocils de la cochlée, par exemple [9]. Chaque groupe de cils a sa fréquence précise et réagit peu aux autres fréquences sonores. Il s'agit d'une propriété vitale pour les systèmes vivants car cela leur permet de décomposer les bruits selon leur fréquence. Les implants cochléaires artificiels ont d'ailleurs plusieurs canaux, chacun représentant une fréquence spécifique. Le défi de ces implants est de relier chaque canal au bon récepteur du patient. Plus il y a de canaux bien connectés, plus la percep-

tion résultante est bonne. Il y a aussi les cellules du cortex somatosenseur qui répondent sélectivement à des régions localisées de la surface du corps. Les cellules d'orientation dans le cortex visuel répondent sélectivement à une stimulation qui est à la fois locale dans sa position rétinale et locale dans l'angle d'orientation de l'objet, tandis que les cellules du noyau laminaire des hiboux sont ajustées à des délais interauraux spécifiques. Les populations de cellules ajustées localement sont typiquement arrangées en cartes corticales où les valeurs auxquelles les cellules répondent varient selon leur position dans la carte (voir figure 0.1). L'ajustement local des stéréocils est une conséquence de leurs propriétés biophysiques. Par contre, dans les cortex la localité de la réponse est une conséquence de l'architecture du réseau. Cette localité est une propriété calculée par le système et ne devrait pas être confondue avec les propriétés de réponse biophysique des cellules.

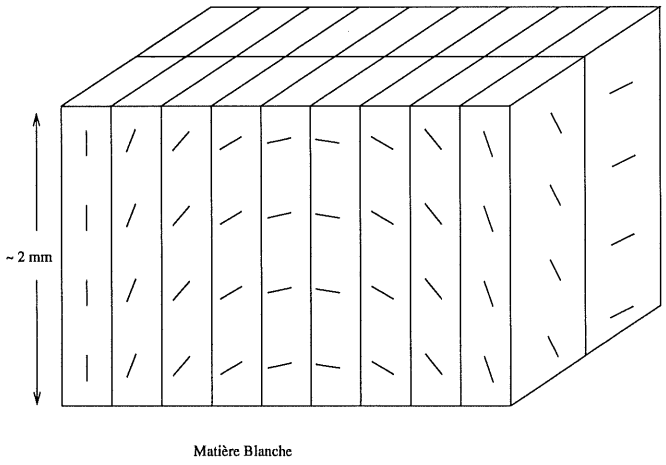

FIG. 0.1 – *Couche de neurones à stimuli local.*
Dans le cortex cérébral, des colonnes de cellules réagissent uniquement à une droite placée dans le champ visuel selon une orientation précise.

Pour obtenir un bon apprentissage dans les réseaux neuronaux multicouches,

il faut que le réseau fasse une optimisation non-linéaire. Ce qui rend le réseau RBF si performant est que l'on préserve le caractère linéaire des calculs afin de garder l'aspect mathématique simple et de minimiser la complexité des calculs [19]. C'est la combinaison de la représentation locale et de l'apprentissage linéaire qui permet au réseau RBF d'offrir de si grands avantages de vitesse comparativement à d'autres réseaux tels le perceptron multicouche.

Le réseau RBF est aussi multimodal, ce qui signifie que l'on peut avoir deux entrées très différentes qui peuvent représenter la même valeur de sortie. C'est un atout pour l'utilisation dans un problème de reconnaissance de caractères manuscrits où chaque symbole peut être dessiné de plusieurs façons différentes. Par exemple, on peut tracer le chiffre zéro avec ou sans diagonale (O ou Ø).

Il a été prouvé [11][12] qu'un réseau RBF peut faire une bonne approximation de toute fonction à plusieurs variables continue sur un domaine compact si un nombre suffisant d'unités est utilisé [8]. Ce nombre est facile à identifier dans la plupart des cas. Malgré cela, il y a quand même certaines limites sur le domaine du problème; certains cas spéciaux de données non linéairement séparables donneront de mauvais résultats. Deux groupes de données sont linéairement séparables si on peut tracer un hyperplan entre les deux groupes. Il peut arriver entre autres qu'un groupe soit inclus entièrement à l'intérieur de l'espace d'un autre groupe; les deux groupes sont donc linéairement inséparables.

Il existe des méthodes de calcul pour la classification et l'interpolation qui ont les mêmes avantages intéressants de parallélisme et de vitesse mais aucune autre ne peut être aussi facilement implantée dans une architecture de réseau

fixe (par exemple, dans des puces électroniques). La simplicité du réseau RBF fait qu'une implantation sous forme de puce électronique serait peu coûteuse.

Nos objectifs sont de présenter le réseau RBF et d'étudier ses performances et capacités. A partir de ces connaissances nous voulons proposer une variante efficace du réseau pour qu'il puisse être implanté dans toute application de façon efficace et économique.

La suite de cet exposé est structurée comme suit. Dans un premier volet nous présenterons les problèmes posés par la tâche de la reconnaissance des caractères manuscrits. Par la suite nous examinerons en détail le réseau RBF et nous examinerons la capacité de ce réseau à faire la classification de caractères manuscrits en fonction des paramètres et ressources qu'il utilise. Dans un deuxième volet nous évaluerons plusieurs techniques d'optimisation et des algorithmes variés qui sont utilisés dans les réseaux RBF afin de faire la reconnaissance de caractères. Cette étude nous amènera à créer une variante du réseau RBF plus simple à utiliser que le réseau de base et plus résistante au bruit.

Afin d'aider à la compréhension et à la comparaison du réseau RBF avec les méthodes statistiques, voici un glossaire des équivalences entre les termes des statistiques et ceux des réseaux neuronaux [19]:

Statistiques	*Réseaux Neuronaux*
modèle	réseau
estimation	apprentissage
régression	apprentissage supervisé
interpolation	généralisation
observations	base d'apprentissage
paramètres	poids synaptiques
variables indépendantes	entrées
variables dépendantes	sorties

TAB. 0.1 – *Équivalences entre les termes provenant des statistiques et des réseaux neuronaux*

La reconnaissance de caractères manuscrits

La reconnaissance des caractères imprimés ou manuscrits par ordinateur est l'objet de recherches continues depuis les vingt-cinq dernières années. L'utilisation de techniques de reconnaissance automatique est en croissance continue; saisie de données par ordinateur portatif sans clavier, tri postal et traitement de formulaires ne sont que quelques-unes des applications de la reconnaissance de caractères.

La saisie de documents dactylographiés présente au départ un défi pour l'informatique. Les résultats dans ce domaine sont intéressants mais jamais parfaits. Malgré cela les entreprises en font grand usage, soit pour faire passer un texte entre des types d'équipement informatique incompatibles ou lorsque

le travail doit être effectué sur des documents provenant de l'extérieur de la compagnie, par exemple.

La reconnaissance de caractères manuscrits représente un défi d'une magnitude supérieure à la saisie de documents dactylographiés; les caractères tracés par un individu peuvent varier selon son état d'esprit, sa fatigue et plusieurs autres facteurs. Entre deux individus, les caractères peuvent varier énormément ou être complètement différents. La difficulté de reconnaître un caractère manuscrit est telle que même l'être humain fait des erreurs, à un taux significatif de près de 4% [13]. Puisque une erreur dans la saisie de données peut avoir des conséquences catastrophiques, il faut s'assurer d'avoir le meilleur taux possible de classement valide des caractères.

Un logiciel de reconnaissance de caractères suit plusieurs étapes lors du traitement. Chacune de ces étapes est nécessaire pour obtenir un résultat et chacune présente des difficultés particulières. Ces étapes visent à transformer l'objet original (un caractère tracé sur une feuille de papier, par exemple) en informations utiles pour l'étape finale de classement du caractère. Les étapes principales pour la reconnaissance de caractères sont généralement les suivantes [22]:

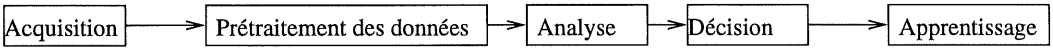

FIG. 0.2 – *Étapes principales pour la reconnaissance de caractères*

L'acquisition

La première étape consiste à obtenir des données numériques à partir de l'objet à traiter. Pour la reconnaissance de caractères, ceci consiste généralement à échantillonner l'image de l'objet en utilisant une caméra vidéo ou un échantillonneur sur feuille de papier. L'image perçue sera transformée en une matrice contenant des valeurs de niveau de couleur (Rouge, Vert et Bleu) ou de niveau de gris.

L'acquisition peut aussi se faire à partir d'une tablette graphique. Dans ce cas l'utilisateur écrit directement sur la tablette. L'ordinateur saisit une image en deux tons du caractère à reconnaître ainsi que de l'information temporelle sur le tracé, ce qui peut aider la reconnaissance de caractères. Un exemple fonctionnel de cette utilisation se retrouve dans le Newton II de la compagnie Apple.

Le prétraitement des données

On reconnaît généralement deux sous-étapes au prétraitement: la localisation et la segmentation. La localisation consiste à rechercher dans l'image saisie l'endroit où se trouve l'information. Elle contribue ainsi à l'élimination des espaces blancs ou du bruit. C'est une étape généralement assez simple pour la reconnaissance de caractères. La segmentation, elle, consiste à séparer les blocs de caractères en lignes, mots et caractères individuels. La plupart des algorithmes de classement de caractères ne classent qu'un caractère à la fois.

Il est donc important de bien pouvoir les séparer.

A ce stade on applique certaines transformations à l'image comme le seuillage, le lissage, la squelettisation, l'élimination du bruit ou la normalisation de la taille de l'image afin de pouvoir passer ces données à l'étape d'analyse.

Le bruit est une complication qui doit toujours être prise en compte lorsque l'on veut faire de la reconnaissance d'images. Un faible niveau de bruit peut facilement induire le logiciel en erreur. Le bruit a deux sources principales: le matériel utilisé (caméra médiocre, ombres sur l'objet, phénomènes d'étirement de l'image, etc.) et l'objet observé lui-même (nature du support, poussière, transparence, etc.). Il est donc important de construire un système de reconnaissance de caractères résistant au bruit. Pour ce faire, on tente d'éliminer le plus de bruit possible à l'intérieur de la phase de prétraitement ainsi qu'à la phase d'analyse ou de classement.

La méthode principale d'élimination du bruit lors du prétraitement est le lissage. Ceci consiste à appliquer un filtre passe-bas à l'image. Ainsi les éléments de l'image inconsistants avec l'information qui nous intéresse sont réduits en importance.

Le seuillage est un autre traitement appliqué à l'image. Il consiste généralement à transformer une image en tons de gris à une image en deux tons (noir et blanc). La procédure est souvent simplement d'examiner si l'histogramme de l'image peut être séparé en deux parties. Tous les tons en deçà du seuil sont transformés en un ton et ceux au-delà, dans l'autre ton.

La squelettisation consiste à tenter de remplacer le trait du caractère par un squelette qui supporte ce trait. Cela réduit la quantité d'information nécessaire pour représenter le caractère mais cela a aussi des inconvénients. Entre autres, il peut y avoir plus d'un squelette valide pour chaque caractère.

L'analyse

Cette étape cherche à fournir à l'étape suivante, la classification, les données susceptibles d'aider à faire le classement. Le choix de ces données et des méthodes utilisées dépend donc de l'algorithme de classification utilisé à la prochaine étape.

On peut cependant distinguer deux grandes catégories de données créées lors de cette étape:

- L'extraction de primitives géométriques: l'étape d'analyse cherchera des formes (segments, splines, etc.) dans l'image. Cette façon de faire s'oriente vers les méthodes de reconnaissance structurelle.
- L'extraction de propriétés topologiques et géométriques: ce sont des mesures de surfaces, de distances, du centre de gravité, de concavités, etc. Cette façon de faire s'applique plus facilement aux algorithmes de classement connexionistes ou statistiques.

La décision

L'étape la plus importante de la reconnaissance de caractères, mais qui serait impossible sans les étapes précédentes, est l'étape de la décision. Les méthodes de prise de décision les plus utilisées sont:

- La méthode du plus proche voisin
- La discrimination fonctionnelle
- Les méthodes structurelles
- Les méthodes statistiques
- Les méthodes connexionnistes

La méthode du plus proche voisin est l'approche la plus simple à comprendre. Il s'agit de représenter chaque caractère par un vecteur de caractéristiques extraites par les étapes précédentes. On définit des vecteurs prototypes pour chaque caractère à reconnaître. Pour faire la classification, on calcule la distance entre le vecteur du caractère et les vecteurs prototypes; on classe le caractère selon le prototype le plus proche. La notion de distance utilisée varie selon les méthodes.

La discrimination fonctionnelle emploie aussi des vecteurs prototypes. Par contre il s'agit ici de découvrir une fonction permettant de séparer les prototypes d'une classe de ceux des autres classes. Les fonctions utilisées sont

linéaires, linéaires par morceaux ou non-linéaires. Le problème avec cette méthode est que les classes ne sont pas toujours linéairement séparables. Dans ce cas, il faut utiliser des fonctions de discriminations non-linéaires, qui sont plus complexes et qui demandent plus de ressources à l'ordinateur.

L'approche structurelle consiste à considérer les objets en fonction de leurs composantes, de leurs propriétés et des relations entre ces composantes. Dans le domaine qui nous intéresse, les objets sont en général des primitives élémentaires. Elles sont assemblées différemment suivant que l'on utilise la théorie des graphes ou la théorie des langages formels.

Dans les représentations issues de la théorie des graphes, les composantes élémentaires sont liées pour une comparaison entre le graphe de la forme et celui du modèle. Dans les représentations en langages formels, la chaîne de primitives est considérée comme un mot d'une grammaire. L'analyse syntaxique détermine si ce mot est correct et correspond à une classe prédéfinie.

De très nombreux systèmes ont été développés avec ces méthodes, surtout en ce qui concerne la reconnaissance des lettres. Les performances de tels systèmes sont intéressantes car elles dépassent les 99% en reconnaissance des caractères latins imprimés et 80% pour les caractères manuscrits.

L'approche statistique la plus couramment utilisée est l'approche bayesienne. Elle a été formalisée par Chow en 1965 [26] et consiste à représenter le problème sous la forme d'un modèle probabiliste. Elle estime la classe d'appartenance d'un prototype avec un minimum d'incertitude et estime le risque de cette

décision.

Dans certaines approches probabilistes, on définit un modèle comme étant un processus aléatoire pouvant changer d'état à tout instant avec une certaine probabilité pour chaque transition. En reconnaissance de caractères, la composante du temps est remplacée par l'ordre d'apparition des lettres ou des chiffres. Le modèle le plus employé dans les approches stochastiques est le modèle de Markov. Des systèmes pour la reconnaissance de l'écriture manuscrite à l'aide de modèles de Markov ont été développés mais des problèmes de temps de calcul et de formalisation théorique limitent pour l'instant les performances de ces applications [24].

L'approche connexionniste est l'approche la plus récente. Elle utilise des réseaux de neurones artificiels comme classificateurs établissant des frontières dans des espaces de dimensions finies. Les réseaux neuronaux permettent de faire un apprentissage avec très peu de spécifications et ils sont relativement simples à mettre en oeuvre.

Nous utilisons cette approche pour faire la reconnaissance de caractères. Basée sur le réseau RBF, cette approche a la particularité de combiner l'approche de la classification automatique et l'approche connexionniste. On obtient ainsi une méthode beaucoup plus puissante et performante.

L'apprentissage

L'objectif de cette étape essentielle d'un système de reconnaissance de caractères est d'acquérir et d'organiser les connaissances sur les classes et les échantillons à traiter. Pour cela il faut former un ensemble représentatif d'échantillons, appelé base d'apprentissage, à partir duquel le système va s'organiser afin de pouvoir faire la classification. On place cette étape à la fin car certains systèmes améliorent leur performance de façon dynamique, c'est-à-dire que l'apprentissage se fait au fur et à mesure que le système fait la classification des données en entrée.

Il existe deux types d'apprentissage: supervisé et non-supervisé. L'apprentissage est supervisé quand un système externe (une personne) présente les échantillons en précisant à quelle classe ils appartiennent. L'apprentissage est dit non-supervisé quand le système recherche de lui-même le nombre de classes présentes et fait la classification parmi ces classes à partir de la base d'apprentissage.

Chaque méthode de décision a sa méthode d'apprentissage associée. Pour la méthode du plus proche voisin, on utilise souvent un apprentissage non-supervisé qui exploite un principe de proximité. La discrimination fonctionnelle emploie un algorithme d'apprentissage supervisé qui permet de calculer les coefficients des différentes fonctions de discrimination [25].

Pour les méthodes structurelles, l'apprentissage implique la construction d'une grammaire à partir de mots ou de phrases convenablement choisis. Le problème

posé par ce type d'apprentissage est que le nombre de grammaires possibles est infini. Cependant, certains algorithmes permettent d'émonder l'arbre de recherche en posant des contraintes sur la grammaire.

L'apprentissage bayesien est un apprentissage supervisé particulier. Il consiste à estimer la probabilité et la distribution de chaque classe. La probabilité peut être soit connue a priori ou estimée facilement à partir de la base d'apprentissage. Pour estimer la distribution, il existe plusieurs méthodes, la plus employée étant celle de l'estimateur de vraisemblance. Pour les systèmes employant des modèles markoviens, l'apprentissage consiste simplement à calculer les probabilités de changement d'état sur un ensemble de prototypes connus.

Finalement, pour les méthodes connexionnistes, l'apprentissage consiste à modifier le poids des connexions entre les neurones en fonction de l'erreur de classification commise par le réseau neuronal. Plusieurs approches connexionnistes ont été utilisées pour la reconnaissance de caractères, entre autres le perceptron multicouche à rétropropagation du gradient d'erreur (feed-forward backpropagation), la machine de Boltzman, les cartes de Kohonen, etc.

Description détaillée de deux méthodes de classification

La classification bayesienne

Cette approche est basée sur le théorème de Bayes appliqué aux distributions de probabilité. Supposons que les caractéristiques X d'un problème soient connues et que l'on veut calculer les probabilités des H sorties possibles mutuellement exclusives qui peuvent être une solution s_h du problème posé:

$$P(s_h|X) = \frac{P(s_h)P(X|s_h)}{\sum_{h=1}^{H} P(s_h)P(X|s_h)}$$

où $P(s_h)$ est la probabilité a priori de s_h, $P(X|s_h)$ est la probabilité conditionnelle de X sachant que s_h est présent et $P(s_h|X)$ est la probabilité conditionnelle de s_h sachant que X est présent.

On veut donc calculer la probabilité a posteriori, soit avec quelle fréquence s_h apparaîtra si les caractéristiques du problème sont X. Pour cela il faut connaître les probabilités $P(s_h)$ et $P(X|s_h)$. Il est généralement impraticable dans un domaine réel de calculer les probabilités $P(X|s_h)$ pour toutes les combinaisons possibles d'un problème. On doit faire l'hypothèse que les N caractéristiques de X sont conditionellement indépendantes. Cela nous permet

de calculer $P(X|s_h)$ avec un plus petit nombre de probabilités:

$$P(X|s_h) = \prod_{n=1}^{N} P(x_n|s_h).$$

Ces méthodes ne sont en général pas satisfaisantes car:

1. certaines caractéristiques peuvent appartenir à deux classes différentes, donc la solution finale a toujours une erreur,
2. les hypothèses faites dans ces méthodes (ex.: que les caractéristiques soient indépendantes) ne sont pas raisonnables en général,
3. ces méthodes utilisent souvent les probabilités conditionnelles $P(s_h|X)$ pour représenter l'aspect probabiliste d'une relation de cause à effet entre la classe s_h et la caractéristique x_n. Ces probabilités conditionnelles, qui ne sont pas des mesures de la force du lien causal, sont imprécises et difficiles à obtenir.

On peut alléger ces problèmes en utilisant une méthode qui spécifie la force d'une association entre une caractéristique et une classe ainsi que les probabilités a priori. Pour cela on peut utiliser la fonction de densité si elle est connue. Quand aucune évidence ne supporte une forme de la fonction de densité plutôt qu'une autre, une bonne solution est souvent de construire une collection de prototypes bien classés appelée base d'apprentissage, et de classer les nouveaux exemplaires en se servant de l'évidence des prototypes les plus près de ceux-ci.

Une de ces procédures non-paramétriques est connue sous le nom de k plus proches voisins votants (k nearest neighbors ou k-NN). Selon cette règle un

exemplaire est classé avec la classe de la majorité de ses k plus proches voisins dans la base d'apprentissage. Le théorème de Cover [23] nous donne une justification statistique de cette procédure en démontrant que lorsque le nombre d'exemplaires N et le nombre de voisins k tendent vers l'infini tel que $k/N \to 0$, le taux d'erreur de la règle $k-$NN s'approche du taux d'erreur de la bayésienne optimale. Par contre dans un problème avec un nombre d'exemplaires fini, la règle $k-$NN n'est pas garantie d'être la façon optimale d'utiliser l'information contenue dans le voisinage. Si on suppose qu'il y a un nombre infini d'exemplaires, l'erreur de classification ϵ_{NN} du classificateur par k plus proches voisins et l'erreur de Bayes ϵ_B suivent la relation:

$$\varepsilon_B \leq \varepsilon_{NN} \leq 2\varepsilon_B - \frac{N}{N-1}\varepsilon_B^2$$

où N est le nombre de classes [30]. En fait le classificateur $k-$NN peut être utilisé pour estimer l'erreur de Bayes dans certaines applications. Le temps d'apprentissage de cette méthode est utilisé pour calculer toutes les probabilités $P(s_h|X)$. Elle nécessite beaucoup d'espace-mémoire pour emmagasiner les exemplaires et un long temps de calcul pour la classification.

Dans les problèmes de classification, une connaissance complète des statistiques concernant les fonctions de densité conditionnelles de chaque classe est rarement disponible, ce qui nous empêche d'utiliser la procédure de classification optimale bayesienne. Si on se sert de k-NN, on peut utiliser la règle de *rejet par distance* pour décider qu'un exemplaire ne fait temporairement partie d'aucune classe et qu'on devrait créer une autre classe car cet exemplaire se trouve loin de tout autre groupe. Cette méthode est multimodale et dynamique, limitée seulement par la mémoire de l'ordinateur.

En résumé, il y a certaines barrières qui nous empêchent d'utiliser la classification bayesienne efficacement. Il faut que la base de connaissances contienne une grande quantité de probabilités. Nous devons aussi supposer des indépendances de façon irréaliste entre les classes. Une autre barrière est l'explosion du nombre de solutions possibles lorsque les caractères à classer possèdent plusieurs caractéristiques similaires, entre autres.

Nous ne discutons ici que de la classification bayesienne de base. Il existe plusieurs modèles récents basés sur la classification bayesienne, tels que l'utilisation de la théorie de Dempster-Shafer sur la règle k-NN, qui tentent de réduire les problèmes associés à ce modèle et s'appliquent entre autres à créer une couverture parcimonieuse du domaine du problème afin d'en simplifier la résolution.

Le réseau perceptron multicouche

Le perceptron multicouche est un réseau neuronal souvent utilisé pour faire de la reconnaissance de caractères. Malgré qu'il ait été très étudié, ce réseau conserve certaines limites intrinsèques. A cause de son aspect global, il nécessite moins de ressources que la classification bayesienne mais par contre il offre des résultats moindres que la classification bayesienne ou que des réseaux neuronaux plus récents tels que le réseau RBF [31].

Des algorithmes bien établis tels que la rétropropagation du gradient d'erreur existent pour entraîner le réseau. C'est un algorithme supervisé qui fonctionne

ainsi: les neurones de la couche de sortie calculent la différence entre la valeur obtenue et la réponse attendue. On ajuste les poids synaptiques en utilisant la règle d'apprentissage de Widrow-Hoff pour les neurones non-linéaires. L'erreur est propagée à reculons à travers les mêmes connexions synaptiques vers la couche cachée. Le même genre de calcul est effectué sur la couche cachée et l'erreur est propagée vers l'entrée. Ainsi l'erreur sera plus petite à l'avenir lorsque cet exemplaire sera présenté en entrée.

Afin de pouvoir enseigner au réseau une classification non-linéaire avec la rétro-propagation du gradient d'erreur, le réseau doit utiliser des neurones non-linéaires. La fonction d'activation non-linéaire la plus populaire est

$$f(x) = \frac{1}{1 + e^{-x}}.$$

La fonction de réponse de chaque couche est

$$\bar{r_k} = f(W\bar{r}_{k-1})$$

où $\bar{r}_{k-1}$ est la réponse de la couche précédente et W est la matrice des poids de connexions.

La règle d'apprentissage pour chaque couche au temps t est

$$W_{(t+1)} = W_{(t)} + \eta\delta_k\bar{r}_k^T$$

L'algorithme de rétropropagation converge vers un minimum local de l'erreur moyenne au carré pour la couche de sortie si la constante η est bien choisie.

La valeur de δ est différente selon la couche où se fait l'apprentissage. Pour la couche de sortie c'est

$$\delta_k = \bar{r}_k(1 - \bar{r}_k)(\bar{o}_k - \bar{r}_k)$$

où $\bar{r}_k$ est la réponse obtenue et $\bar{o}_k$ la réponse attendue. Pour la couche cachée on ne connaît pas la réponse attendue. On se sert plutôt de l'erreur rétropropagée à partir de la couche de sortie pour estimer l'erreur à cette couche-ci:

$$\delta_k = \bar{r}_k(1 - \bar{r}_k)(W^T \delta_{k+1})$$

où δ_{k+1} est le vecteur obtenu à la sortie et rétropropagé.

Un des désavantages de ce réseau est qu'en général il nécessite un grand nombre d'itérations pour converger vers une solution. En effet, il faut que l'apprentissage se fasse sur tous les paramètres en même temps et cela requiert beaucoup d'itérations pour que chaque paramètre soit bien ajusté. Chaque itération demande un grand nombre de calculs, plus que le réseau RBF.

De plus, il est très difficile d'estimer la quantité de neurones nécessaires pour avoir une architecture optimale. Très souvent les utilisateurs tentent de deviner la quantité nécessaire, examinent le résultat de l'apprentissage avec cette quantité et recommencent tant qu'ils ne sont pas satisfaits du résultat.

Ce n'est donc pas un réseau aussi simple à utiliser qu'il y paraît. Néanmoins, il demeure fiable et un utilisateur averti peut obtenir de très bons résultats.

Chapitre 1

Présentation du Réseau RBF

1.1 Description détaillée du réseau

Ce réseau est utilisé pour trouver une approximation d'une fonction non-linéaire ainsi que pour faire l'interpolation d'une fonction définie sur un ensemble fini de points. L'idée principale est de construire une transformation non-linéaire entre l'espace d'entrée et l'espace de sortie par un processus à deux étapes. La première étape est une simple projection non-linéaire de la couche d'entrée à la couche cachée. La deuxième étape établit une transformation linéaire de la couche cachée vers la couche de sortie.

Dans sa forme la plus simple l'architecture du réseau est prédéterminée. En théorie le vecteur en entrée est passé à la couche cachée grâce à des poids de

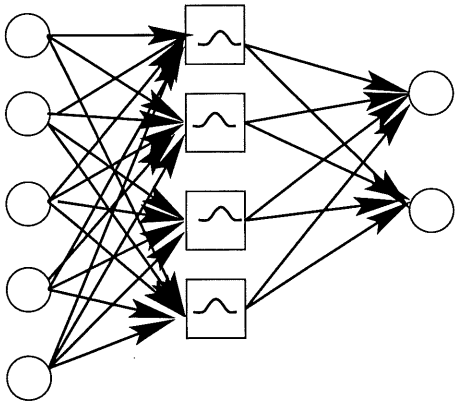

FIG. 1.1 – *Architecture du réseau.*

La couche de gauche est la couche d'entrée. Le vecteur entré passe à la couche cachée par des connections de poids unitaires. Le résultat des fonctions gaussiennes de la couche cachée est passé à la couche de sortie avec les poids de connection de la matrice W.

valeur unité. En pratique on ignore simplement cette couche. Chaque neurone caché du réseau représente un groupe grâce à une fonction radiale centrée sur ce groupe. On se sert de l'algorithme des k-Moyennes pour déterminer les groupes; c'est pourquoi on dit que le réseau RBF est de type assisté par regroupement (clustering-aided type). La valeur du rayon de chaque fonction radiale est déterminée par différentes méthodes, entre autres en utilisant la moyenne des distances des k plus proches voisins . Toutes les fonctions radiales de base dans la couche cachée sont du même type.

Le réseau RBF peut utiliser un ou plusieurs neurones de sortie. Nous avons utilisé 10 neurones de sortie, un pour chaque chiffre de 0 à 9, encodé dans le style *un-de-x*. Cela signifie qu'on donne une valeur numérique aux sorties dans la base d'apprentissage selon que l'exemplaire a une probabilité 1 d'ap-

partenir à la classe s et 0 d'appartenir à toute autre. Ainsi les vecteurs sortie dans l'apprentissage sont des vecteurs de grandeur égale au nombre de classes et qui contiennent un seul 1 à la position s et des 0 partout ailleurs. Après l'apprentissage, le réseau répond à un nouvel exemplaire avec des valeurs continues dans le vecteur sortie qui sont interprétées comme proportionnelles à la probabilité d'appartenance à chaque classe. Un neurone de la couche de sortie s'activera plus fortement pour signaler l'appartenance du vecteur d'entrée à la classe que ce neurone représente.

Le cas général est que les entrées et les sorties sont des vecteurs. Ceci rend la notation plus complexe sans aider à la compréhension. Aussi dans la littérature sur ce réseau, la notation généralement utilisée ne représente qu'un seul neurone de sortie afin de simplifier la notation. Nous adoptons aussi ce principe mais avec une notation plus détaillée lorsque que l'explication la requiert.

Soit $X = \{x_1, x_2, \ldots, x_N\}$ l'ensemble des exemplaires pour l'apprentissage.

$E = \{e_1, e_2, \ldots, e_F\}$ l'ensemble des neurones d'entrée.

$C = \{c_1, c_2, \ldots, c_K\}$ est l'ensemble des neurones cachés qui ont chacun un rayon σ_k associé. Pour simplifier, les c_k représentent également le centre du neurone c_k.

$S = \{s_1, s_2, \ldots, s_H\}$ est l'ensemble des neurones de sortie.

Plus spécifiquement, le problème général est d'approximer une fonction g de

R^F vers R^H définie sur N observations par une fonction f telle que

$$g(x) \approx f(x) \text{ pour } x \in X. \tag{1.1}$$

L'idée générale est de représenter la fonction f par une somme de fonctions (habituellement non-linéaires) ϕ_k telle que l'équation 1.1 est approximée par

$$g(x) \approx \sum_{k=1}^{K} w_k \phi_k(x, \lambda_k) \tag{1.2}$$

où w_k sont des vecteurs de poids synaptiques et λ_k est un ensemble de paramètres à préciser selon le contexte. Dans ce travail, λ_k est la paire (c_k, σ_k), soit le centre et le rayon du neurone c_k.

On pourrait en théorie se servir de toutes les entrées x_l pour simuler la fonction f, mais en pratique ce n'est pas désirable, car il faudrait poser un centre ϕ pour chaque x. On cherche plutôt à faire l'approximation de la fonction g par le plus petit nombre possible de fonctions ϕ où chaque ϕ peut représenter un groupe d'exemplaires de X. Cette technique porte le nom d'approximation par fonctions radiales de base lorsque l'on utilise plusieurs groupes plutôt que X directement et que la distance entre le vecteur x_l et un centre est utilisé plutôt que la position d'un centre (qui remplace x_l) directement, dans l'équation 1.2. On les nomme fonctions de base par analogie avec le concept d'un vecteur obtenu comme combinaison linéaire de vecteurs de base.

Les choix typiques de ces fonctions radiales de base ϕ sont:

spline de plaque mince: $\phi(v) = v^2 * log(v)$

Gaussienne: $\phi(v) = e^{-(v^2/\sigma^2)}$

multiquadrique: $\phi(v) = (v^2 + \sigma^2)^{1/2}$

multiquadrique inverse: $\phi(v) = \frac{1}{(v^2+\sigma^2)^{1/2}}$

Cauchy: $\phi(v) = (1 + v)^{-1}$

où v est la valeur non-négative de la distance entre le vecteur entré x et le centre c et σ est le rayon des fonctions radiale de base. Les gaussiennes sont les plus utilisées pour leur aspect local, alors que les fonctions multiquadriques le sont peu car elles ont une réponse globale.

La méthode utilisée par le réseau RBF pour représenter les entrées de façon locale est de les grouper dans un espace à I dimensions, où I = nombre de caractéristiques en entrée. Chaque groupe possède un centre et un rayon qui sont déterminés lors de l'apprentissage. Un vecteur en entrée x sera comparé à tous les centres et sera classé avec celui qui minimise la distance $d = ||x - c_k||$ où $||\cdot||$ est la distance euclidienne (ou distance de Mahalanobis [17]). La couche de sortie s'occupe d'associer chaque groupe avec la classe appropriée.

L'apprentissage du réseau est un problème d'estimation et ici la solution peut être obtenue avec la résolution d'un ensemble d'équations linéaires. La complexité augmente avec la quantité d'exemplaires ce qui rend la résolution de ces équations irréalisable. Pour des applications pratiques il est nécessaire d'utiliser un réseau avec un nombre limité de fonctions de base. C'est pour cette

raison que l'on doit faire appel à des méthodes d'apprentissages plus complexes comme le regroupement auto-organisés par l'algorithme des k-moyennes.

L'algorithme k-Moyennes

Dans les problèmes de classification, on possède rarement des statistiques complètes sur la fonction de densité de chaque classe, ce qui nous empêche d'utiliser une procédure de classification Bayes-optimale. Quand aucune évidence ne suggère une forme de fonction de densité plus qu'une autre, une bonne solution est souvent de construire une collection d'exemplaires classés correctement, nommée la base d'apprentissage, et de classer chaque nouvelle entrée en utilisant l'évidence fournie par les exemplaires les plus proches. Il est très important de trouver un ensemble de vecteurs représentatifs pour les applications de classification.

On dit que les groupes sont représentatifs lorsque l'on peut partitionner les x_i exemplaires en K ensembles $[G_1, \ldots, G_K]$ où chaque ensemble G_k correspond aux points de X à l'intérieur du polytope de Voronoi de centre c_k:

$$G_l = \{x_i : ||x_i - c_k|| < ||x_i - c_j||, j \neq k\}$$

où $||\cdot||$ est la norme euclidienne. L'ensemble des groupes est optimal s'ils minimisent la mesure d'erreur

$$\varepsilon = 1/2 \sum_{i=1}^{N} \sum_{k=1}^{K} ||x_i - c_k||^2 1_{G_k}(x_i)$$

où $1_G(\cdot)$ est l'indicateur d'appartenance à l'ensemble G [16].

Une technique utilisée couramment pour trouver un ensemble localement optimal de centres est la suivante. On commence par choisir une configuration de centres aléatoire $m^{(\tau=0)}$. Il faut s'assurer qu'il n'y ait pas deux centres à la même position au départ, mais l'algorithme de base ne prescrit pas une distance minimale entre chaque centre, car on ne peut connaître cette distance à priori. Ensuite pour chaque x_i choisi aléatoirement au temps τ on calcule:

$$\begin{aligned} c_k^{(\tau+1)} &= c_k^{(\tau)} + \eta_\tau (x_i - c_k^{(\tau)}) \text{ si } x_i \in G_k^{(\tau)} \\ &= c_k^{(\tau)}, \text{sinon.} \end{aligned}$$

où $\{\eta_\tau\}$ est une séquence de scalaires non-croissante et c_k est le plus proche voisin de x_i.

L'approche classique de l'algorithme $k-$NN définit une distance appropriée entre les exemplaires X et les centres C et trouve parmi ces centres celui qui est le plus proches que l'exemplaire examiné. On pourrait aussi faire entrer un plus grand nombre de voisins dans le calcul en faisant diminuer leur apport proportionellement à leur distance de l'exemplaire. On peut ainsi décider à quelle classe se rattache l'exemplaire. Cette façon de faire se nomme méthode des k plus proches voisins. Dans notre algorithme, on simplifie les choses en utilisant seulement le plus proche voisin. Lorsque $k = 1$, on associe le groupe à l'exemplaire le plus près. On peut généraliser pour $k > 1$ en utilisant par exemple un vote sur la classe majoritaire parmi les classes des k plus proches voisins.

La position des centres à la fin de l'exécution n'est pas la même que celle des exemplaires de X. Cet algorithme est connu sous le nom d'algorithme

des k-moyennes local (Local K-means Algorithm, LKMA). Il a l'avantage de pouvoir fonctionner dynamiquement et les groupes sont auto-organisés. Par contre si la distribution spatiale des données n'est pas uniforme il peut être peu performant.

Une généralisation simple de cette technique est utilisée en classification: pour chaque x_i on a une classe S_{x_i} associée. On peut trouver les centres optimaux pour chaque classe S_k avec:

$$\begin{aligned} c_{S_k}^{(\tau+1)} &= c_{S_k}^{(\tau)} + \eta_t(x_i - c_{S_k}^{(\tau)}), \text{si } x_i \in G_{S_k}^{(\tau)} \text{ et } S(x_i) = S_k \\ &= c_{S_k}^{(\tau)}, \text{sinon.} \end{aligned} \tag{1.3}$$

Cette procédure est similaire à la méthode LVQ1 de Kohonen sauf que LVQ1 utilise aussi un terme de répulsion pour les points élements de G_{S_k} dont la classe n'est pas S_k pour éloigner les centres des bordures de décision. Malgré que LVQ1 fonctionne en général, elle est relativement inefficace car la plupart des groupes sont inutiles pour la classification; seuls ceux qui sont près de la bordure jouent un rôle important.

L'apprentissage pour la couche de sortie

L'étape suivante se fait aussi hors-ligne, mais pourrait être dynamique, car on utilise une simple couche de perceptrons avec règle d'apprentissage Delta (aussi nommée Widrow-Hoff). Pour faire l'apprentissage dans la couche de sortie, nous utilisons les résultats des fonctions radiales de base. On utilise en général la fonction d'activation sigmoïde sur la somme des valeurs d'activation ou directement cette somme pour calculer la valeur de sortie.

Dans la règle Delta, on calcule la différence entre la réponse et le résultat attendu et on modifie les poids en conséquence:

$$w_{i,h}^{(t+1)} = w_{i,h}^{(t)} + \alpha(r_h - o_h)x_i^T$$

où $W^{(t)}$ est la matrice des poids au temps t, α est une constante positive petite et i est choisi aléatoirement. r_h est le vecteur de résultat attendu et o_h est le vecteur résultat obtenu. Cet algorithme converga si α est bien choisi.

On doit utiliser la version normalisée de la réponse du réseau pour simplifier l'apprentissage de la couche sortie. La forme normalisée que nous avons utilisé est

$$g(x) = \frac{\sum_k w_k \exp\left[-\frac{\|x-c_k\|^2}{(\sigma_k)^2}\right]}{\sum_k \exp\left[-\frac{\|x-c_k\|^2}{(\sigma_k)^2}\right]}.$$

Cela simplifie le calcul des poids pour la couche de sortie car les réponses sont toutes proportionnelles donc on n'a pas à ajuster les amplitudes en conséquence.

Il n'y a pas de critère précis pour arrêter l'apprentissage. On peut simplement l'arrêter lorsque l'erreur MSE se stabilise à un minimum local.

L'algorithme d'apprentissage de RBF est une minimisation de l'erreur moyenne au carré, comme dans bon nombre de réseaux neuronaux. Si nous avons assez d'exemplaires et que nous connaissons toujours la sortie attendue, nous pouvons calculer l'erreur moyenne au carré comme:

$$MSE = \langle (s_l - r_l)^2 \rangle$$

où l'espérance $\langle \cdots \rangle$ est calculée sur l'ensemble d'apprentissage, s_l est le résultat attendu et r_l le résultat obtenu. L'erreur moyenne au carré MSE à minimiser

entre les N exemplaires x_n et le centre le plus près c_k est donc:

$$MSE = \frac{1}{2}\sum_{l=1}^{N}(g(x_l) - \sum_{k=1}^{K} w_k \phi_k(||x_l - c_k||))^2. \tag{1.4}$$

où $g(x_l)$ est la sortie attendue pour l'exemplaire x_l.

La minimisation de l'erreur MSE par tous les paramètres (σ_k, c_k, w_k) simultanément est un problème difficile parce que c'est un problème d'optimisation non-linéaire. Lorsque l'on fait l'apprentissage dans un perceptron multicouche à rétropropagation du gradient, on tente d'optimiser les valeurs pour les connexions et les seuils des deux couches en même temps. Cette optimisation est complexe et non-linéaire. Dans le modèle RBF, on cherche à optimiser les centres et les rayons dans la couche d'entrée de façon linéaire auto-organisée et ensuite on se sert de l'erreur MSE pour calculer les poids dans la couche de sortie. Ainsi les valeurs de σ_k sont prédéterminées et les valeurs c_k sont basées directement sur l'ensemble des exemplaires, donc la minimisation peut être simplifiée considérablement parce qu'elle se fait seulement sur les poids w_k. Même si ces neurones utilisent des fonctions quadratiques, la recherche des poids optimaux se fait de façon linéaire. Les calculs lors de l'apprentissage sont donc beaucoup plus rapides que pour la rétropropagation du gradient non-linéaire.

Lorsque cette partie de l'algorithme est terminée, on ne peut plus modifier la position des centres car, si on veut ajouter un exemplaire, il faut recommencer les calculs. C'est pourquoi on dit que cette partie de l'algorithme n'est pas dynamique et se fait hors-ligne.

Sélection des rayons σ

Le choix des rayons σ influence fortement la qualité de la généralisation. Dans la section 1.1 on indique qu'il faut choisir les valeurs σ séparément du reste de l'algorithme. On peut comparer les estimateurs de noyaux de régression (Kernel regression estimators, KRE) à des réseaux RBF et les résultats théoriques sur KRE peuvent être appliqués à RBF [7]. Une analyse mathématique de KRE montre que le choix des valeurs σ est rendu difficile par le fait que la densité de la fonction de distribution n'est pas connue à priori et donc il faut s'en tenir à des méthodes heuristiques ou à des tests concrets pour déterminer les valeurs appropriées des σ. Par contre, les développements mathématiques nous montrent que deux valeurs importantes influencent le choix de la taille des fonctions radiales de base: plus le nombre de fonctions de base est grand, plus on peut réduire la taille du champ réceptif de chacune. Aussi, plus la fonction à approcher est lisse, plus le champ réceptif peut (et doit) être grand.

On peut considérer la valeur σ comme la variation sur chaque groupe. L'heuristique du choix de σ le fait de façon à obtenir un certain degré de couverture entre les champs réceptifs de chaque unité afin d'obtenir une interpolation douce et continue sur les régions du domaine qu'elles représentent. Une méthode heuristique simple est de prendre 3/5 de la distance du plus proche voisin comme champ réceptif.

1.2 Analyse du réseau

Problèmes

Cet algorithme est essentiellement une méthode de descente, au sens où on positionne les unités cachées de façon à diminuer l'erreur à chaque étape. Ce processus de réduction d'erreur est local et le réseau final peut être piégé dans un minimum local. De plus, le réseau tend facilement vers le surapprentissage (overfitting) des données d'entraînement. S'il y a des unités cachées sous-utilisées, celles-ci formeront des groupes avec des exemplaires aberrants. Le surapprentissage est un phénomène qui diminue la qualité de la classification parce que le réseau devient trop bien adapté au éléments de la fonction à apprendre qui sont dans la base d'apprentissage. Lorsque le réseau est trop bien optimisé pour ces données il perd de sa capacité à généraliser sur les données qui ne sont pas dans la base d'apprentissage.

Un autre désavantage est qu'il faut mettre toutes les données en entrée à la même échelle, sinon le calcul du rayon ne fonctionnera pas. D'autres versions plus complexes de l'algorithme utilisent un rayon par dimension par centre.

Ce modèle n'est pas un bon choix pour apprendre des projections logiques telles la parité, parce que de telles projections ne sont pas continues mais la sortie du réseau l'est.

Un des désavantages principaux de l'algorithme k-moyennes est qu'on assume

implicitement que le voisinage d'un exemplaire x est contenu dans une région de volume relativement faible. Il faudrait utiliser un autre algorithme si ce n'est pas le cas. Une autre limite est que k-moyennes n'offre aucune façon de traiter l'incertitude ou l'imprécision des classes de la base d'apprentissage.

Le réseau RBF est multimodal. Cela signifie que pour tous les centres différents mais de la même classe, il faut que la couche de sortie fournisse la même réponse. Par contre l'hétéro-associateur linéaire de la couche sortie limite le nombre centres qui peuvent être utilisés dans la couche cachée. D'un point de vue matriciel, il y a une limite au nombre de vecteurs différents qu'un nombre de neurones fixe peut apprendre sans erreur. Il nous faut environs $\sqrt{NH}$ neurones pour stocker N vecteurs d'entrée et H vecteurs de sorties orthogonaux. Pour contrer ce problème, on peut faire appel à plusieurs techniques comme par exemple rajouter une couche de neurone intermédiaire ou bien tenter le plus possible que les vecteurs d'appartenance soient orthogonaux.

Avantages

Comparativement à des réseaux multicouche à rétropropagation du gradient, il y a l'avantage de la vitesse. Le réseau RBF utilise deux couches avec des fonctions de calcul linéaires. Dans des réseaux à rétropropagation du gradient, il y a 2 couches ou plus qui utilisent des fonctions de calcul logistiques. Le calcul de ces fonctions mathématiques est relativement long donc le réseau RBF est avantagé. De plus, contrairement au perceptron multicouche, l'apprentissage de RBF ne fait pas varier tous les paramètres en même temps, ce qui facilite

l'apprentissage des paramètres non fixés d'avance et diminue la complexité des calculs.

L'algorithme RBF de base est très simple et facilement modifiable. Il possède de bonnes capacités de généralisation; il garantit la convergence, mais pas nécessairement vers la solution optimale. Des résultats expérimentaux montrent que cet algorithme est robuste contre le bruit [5]. Contrairement au réseau probabiliste, on n'a pas à garder tous les exemplaires en mémoire mais seulement la position et le rayon de chaque groupe.

L'algorithme RBF est capable de faire la séparation linéaire dans la plupart des cas, sauf lorsque les données d'un groupe x_1 sont partiellement immergées dans le groupe x_2. (voir la figure 1.2). Le réseau ne peut former un groupe à l'intérieur de l'autre groupe de façon efficace car l'heuristique du calcul de rayon fait que si on utilise deux groupes, les rayons de chaque groupe seront diminués dramatiquement et donc que plusieurs exemplaires seront à l'extérieur de leur groupe. Si on utilise un seul groupe, évidemment on ignore l'autre groupe complètement. Nous verrons plus tard que l'on peut améliorer cet aspect de l'algorithme, par exemple en augmentant le nombre de centres.

La localité des fonctions radiales est importante pour optimiser les calculs. Pour toute entrée, seulement une petite fraction des processeurs de fonctions radiales près des entrées aura une réponse qui diffère de zéro de façon significative. Ainsi, seuls les processeurs avec des centres assez près de l'entrée doivent être évalués et entrainés.

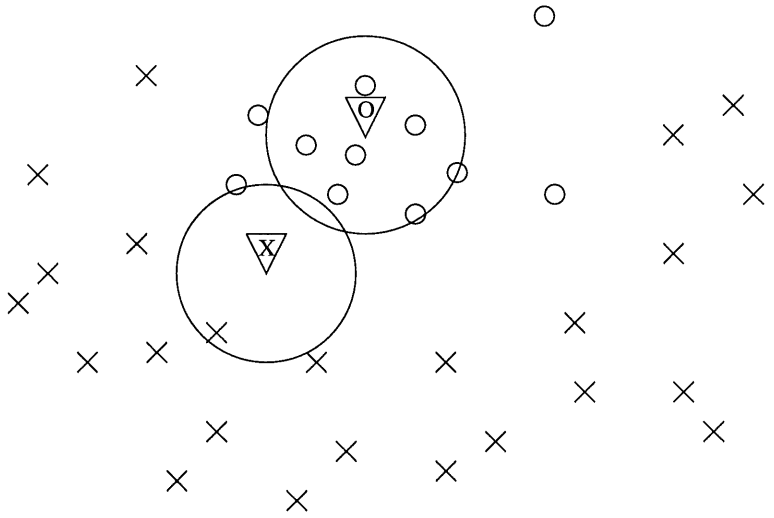

FIG. 1.2 – *Exemple de classes immergées.*

Le centre de chaque groupe est indiqué par un triangle. On voit que le groupe X contient peu d'élément car le groupe O est trop près.

Chapitre 2

Données et expérimentation

2.1 Données utilisées

Toutes nos expériences sont basées sur une des bases de données du National Institute for Standards and Technology (NIST). La base que nous utilisons est distribuée sous le nom de *fl3*. Elle est un sous-ensemble, distribué gratuitement, de la base de données principale du NIST (NIST Special Database 1). Ce sous-ensemble ne porte que sur les chiffres manuscrits 0 à 9 écrits par 49 personnes différentes, plusieurs fois. La base *fl3* contient 3471 exemplaires au total et chaque chiffre s'y retrouve avec environ la même probabilité de 10%. La base de données complète originale comprend en plus des chiffres, l'alphabet de a à z pour un total de plus de un million de caractères. Le fait que cette base soit distribuée gratuitement et que le prétraitement soit déjà fait l'a

rendue assez populaire parmi les chercheurs en reconnaissance de caractères et plusieurs articles s'y réfèrent [31][30][15]. Ceci permet de faire rapidement une comparaison directe entre les performances de plusieurs méthodes.

Tous les caractères sont déjà de taille normalisée dans la base NIST et sont représentés dans une matrice de 32 par 32 pixels de valeur 0 ou 255. Les images sont centrées, découpées et tout le prétraitement a déjà été fait par le NIST. Pour obtenir ses données le NIST a prédéfini des formulaires avec des cadres où un certain nombre de caractères choisis d'avance devaient être inscrits. L'aquisition s'est faite en noir et blanc à 300 pixels par pouce, ce qui donne des images de pages de caractères de 2560 par 3296 pixels. Pour faire la localisation il a suffit de retrouver la position moyenne de ces cadres sur l'image acquise. Le processus de segmentation a été amélioré en comparant le nombre de caractères attendus avec le nombre obtenu par la segmentation. Ensuite la taille de tous les caractères à été normalisée en se basant sur la taille standard des cadres. Les caractères saisis ont de plus subis une légère rotation pour corriger la rotation induite lors de la reproduction et la saisie du formulaire, encore une fois en se basant sur l'image d'un cadre standard.

Nous avons utilisé la forme de base de validation croisée, soit la division de l'ensemble des exemplaires en base d'apprentissage et en base de test, soit 1157 exemplaires dans la base d'apprentissage et 2314 exemplaires dans la base de tests. Une meilleure méthode, qui tente d'éviter tout biais introduit par l'utilisation d'une division quelconque de l'ensemble des exemplaires, est de partitioner l'ensemble de plusieurs façons différentes et de calculer une valeur moyenne de l'erreur sur toutes les partitions. Cependant, cette méthode

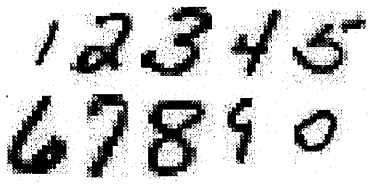

FIG. 2.1 – *Exemples de caractères de la base fl3 choisis au hasard.*

demande beaucoup plus de manipulations et de temps de calcul et il est assez rare qu'on y fasse appel en pratique. Pour le logiciel de reconnaissance de caractères, les données sont représentées comme suit:

- *Vecteur d'entrée:* vecteur de 256 nombres de $[0, 1]$ en double précision représentant le sous-échantillonnage 16x16 d'un chiffre manuscrit. Le nombre de pixels de chaque image est donc divisé par 4.

- Les neurones cachés prennent ces vecteurs en entrée et retournent des *vecteurs d'appartenance*: vecteurs de 256 éléments de valeur $[0, 1]$ en double précision. Il y a création d'un vecteur d'appartenance pour chaque groupe.

- Ce vecteur d'appartenance est passé à la couche de sortie. Celle-ci créée le *vecteur de sortie*: vecteur avec H éléments de $[0, 1]$ en double précision. L'élément s_h du vecteur de sortie s prends sa valeur maximum lorsque le vecteur entré appartient avec certitude à la classe h. Comme nous voulons classer les chiffres 0 à 9, on pose le nombre de classes $H = 10$.

- On peut définir un critère d'ambiguité, par exemple $\forall s_h < 0.2$, si le réseau ne peut classer correctement. Si toutes les sorties s_h sont de valeur négligeable, alors le réseau n'a pas reçu un entraînement valide et ne peut généraliser correctement ou le vecteur entré ne ressemble à aucun prototype.

2.2 Implantation

Le réseau utilisé est le modèle de base sans modification. On se servira de la fonction d'activation gaussienne avec un seul rayon par centre pour toutes les dimensions du domaine du problème. L'utilisation d'un seul rayon par centre simplifie l'algorithme. On peut utiliser un rayon par dimension du domaine du problème mais il n'est pas démontré que cela améliore les performances du réseau.

Nous voulons séparer les données en 10 classes indépendantes. Nous avons donc besoin de 10 neurones de sortie. Par contre on ne peut pas savoir à priori combien de groupes seront nécessaires. L'expérience nous a montré que l'utilisation de 120 centres donne un bon résultat avec notre base d'apprentissage (voir 2.4). Le nombre de poids synaptiques qui doivent être ajustés est donc environs 31920.

Le matériel utilisé est un ordinateur avec processeur Pentium à 100 MHz avec 16 megaoctets de mémoire, sous le système d'exploitation Linux. Le logiciel a

été rédigé en C et compilé avec Gnu C++ 2.7.2. Les bases de données sont des fichiers de nombres représentés par des caractères ASCII.

2.3 Présentation des expérimentations

Nous cherchons à estimer les performances générales du réseau ainsi qu'à déterminer quelles sont ses faiblesses au niveau pratique. Les tests suivants couvrent la plupart des informations que l'on peut retirer directement du réseau:

1. Expérimentation pour déterminer les valeurs de σ, α, η, le nombre de centres et le nombre d'itérations optimales.
2. Taux d'erreur de la classification en fonction du nombre d'itérations.
3. Taux d'erreur de la classification en fonction du nombre de groupes.
4. Taux d'erreur de la classification de la base d'apprentissage.
5. Taux d'erreur lors de la généralisation.
6. Taux d'erreur de la classification de la base d'apprentissage bruitée.
7. Taux d'erreur lors de la généralisation de la base de tests bruitée.

Les tests 1 à 3 servent à déterminer l'architecture optimale du réseau. Les valeurs de ces paramètres dépendent souvent des données du problème et sont difficiles à estimer à priori. On ne fait varier qu'une des constantes du réseau

à la fois, par exemple le nombre de centres ou le nombre d'itérations pour l'apprentissage et on note le taux de classifications correctes obtenu. Ensuite on recommence l'apprentissage du début avec une nouvelle valeur pour la constante. Ainsi on obtient assez de statistiques pour estimer quelles sont les valeurs optimales des constantes d'apprentissage. Ces tests utilisent les deux bases de données.

Les valeurs que l'on cherche ainsi sont σ, qui est le rayon de chaque groupe, η qui est le taux d'apprentissage des vecteurs et α qui est le taux d'apprentissage des poids du réseau.

Il faut utiliser un facteur, m_σ, pour diviser le rayon calculé comme la distance entre un centre et son plus proche voisin. Le facteur m_σ peut prendre une valeur entre $[0.5, 1[$. Un $m_\sigma = 0.5$ est la valeur minimale car il s'agit de la moitié de la distance entre un centre et son plus proche voisin, si on utilise un m_σ plus petit alors les exemplaires entre les deux centres risque de ne pas tomber facilement ni dans l'un ni dans l'autre. On peut utiliser une valeur de m_σ plus grande que 0.5 pour améliorer la couverture de l'espace par les centres. Les constantes η et α peuvent prendre n'importe quelle valeur dans $[0, 1]$.

Les tests sur la base d'apprentissage 4 et 6 servent à vérifier si le réseau fait effectivement un bon apprentissage. Si après avoir montré la base d'apprentissage au réseau, il peut bien la classer, c'est que l'apprentissage a bien fonctionné.

Les tests 5 et 7 nous permettent de voir la performance du réseau pour la généralisation. Si le résultat est bon, c'est que le réseau a appris une bonne transformation générale du domaine du problème vers son image.

Les tests 6 et 7 nous montrent comment le réseau réagit au bruit. Le bruit étant une composante importante des problèmes de reconnaissance de caractères, il est important de fournir des tests appropriés. Nous avons donc ajouté un bruit dont la fréquence et l'amplitude suivent la distribution de la Normale. Cette façon de faire ne simule pas de manière réaliste tous les types de bruits que l'on peut rencontrer lors de la reconaissance de caractères mais nous permet tout de même de voir si le réseau résiste au bruit.

2.4 Résultats obtenus

Test 1. Déterminer les valeurs des constantes d'apprentissage

Nous avons commencé par tenter de découvrir les meilleures valeurs de paramètres à utiliser. Pour cela nous utilisons un simple procédé par essais et erreurs. Nous avons généré une solution avec un ensemble de données en ne changeant que la valeur d'un paramètre à la fois. Nous gardons les valeurs qui ont données les meilleures solutions comme étant les meilleures valeurs pour ces paramètres.

Nous avons ainsi trouvé que la meilleure valeur pour m_σ est 0.8. Dans la

littérature sur ce modèle on nous propose généralement de choisir une valeur de 3/5 pour offrir un certain recouvrement entre les groupes. Malgré que cette valeur soit raisonnable, elle dépend beaucoup des données du problème. Dans nos données nous avons des groupes assez distants les uns des autres et donc la valeur de σ peut être plus élevée. En effet nous obtenons un classement de 83.7% plutôt que 89.36% si on utilise $m_\sigma = 0.6$ plutôt que 0.8.

Les meilleures valeurs obtenues pour η et α sont $0,3$ et $0,8$ respectivement. Nous n'avons pas utilisé de momentum dans l'algorithme de base. Il faut s'assurer d'arrêter l'exécution du programme avant de faire du surapprentissage.

Aprés avoir ajustés nos paramètres correctement on obtient un taux de classement correct de 89.36% sur la base d'apprentissage et de 81.57% sur la base de tests. Ces résultats sont obtenus sur un réseau avec 120 neurones cachés et 15500 itérations pour l'apprentissage avec les valeurs $\sigma = 0.8$, $\eta = 0.3$ et $\alpha = 0.8$. Cet apprentissage requiert 327 secondes et le rappel, 13 secondes, soit 0.005 seconde par exemplaire lors du rappel.

Test 2. Taux d'erreurs en fonction du nombre d'itérations

A la figure 2.2 on voit que la qualité de la classification atteint un sommet à 15500 itérations puis redescend. Passé 16000 itérations, le résultat devient moins bon dû au surapprentissage. La qualité de la classification continue de s'améliorer légèrement car les résultats ambigus sont finalement résolus, mais cette amélioration est limitée par le surapprentissage.

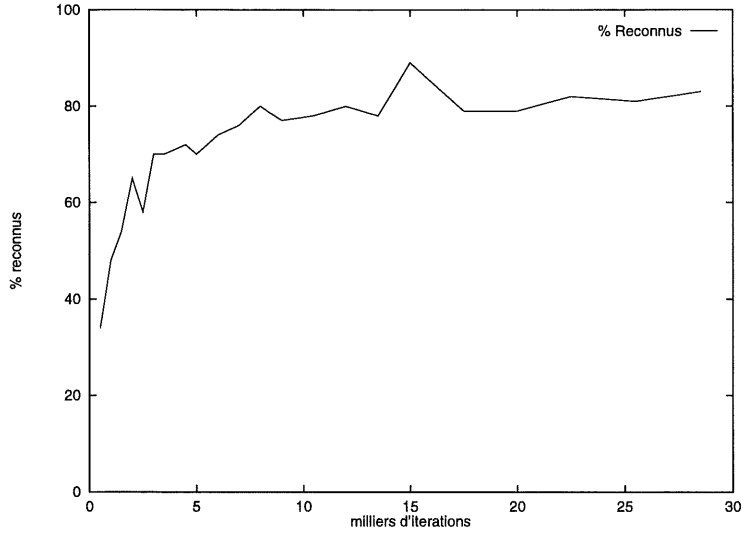

FIG. 2.2 – *Test 2. Taux de classification en fonction des itérations sur la base d'apprentissage.*

Test 3. Taux de classification en fonction du nombre de groupes

On voit à la figure 2.3 que la qualité de la classification augmente en fonction du nombre de centres, comme la théorie le prédit. En effet on sait que l'on tends vers une classification parfaite si on utilise un nombre de centres infinis. Mais la figure 2.4 nous montre que le temps d'apprentissage croît linéairement avec le nombre de centres, donc il faut trouver un juste milieu. Nous avons donc choisi 120 centres.

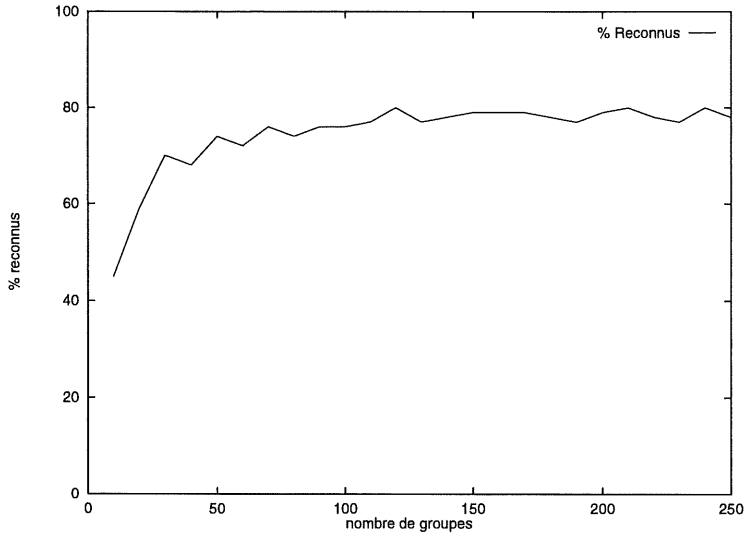

FIG. 2.3 – *Test 3. Taux de classification en fonction du nombre de centres sur la base d'apprentissage.*

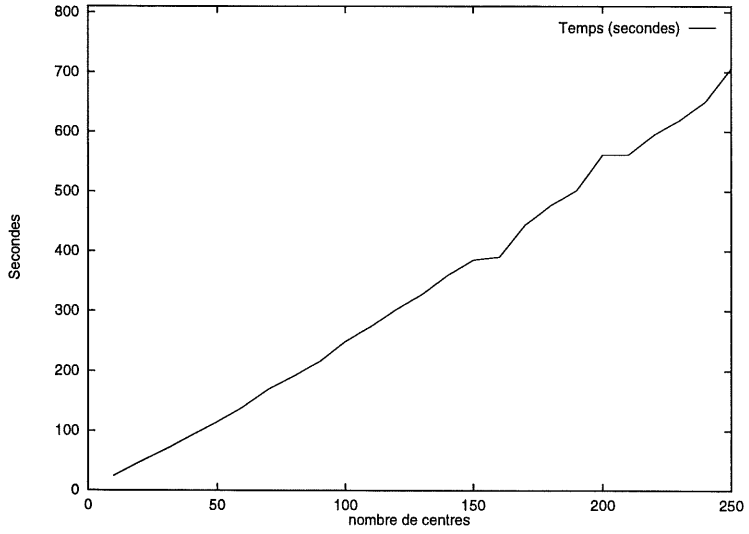

FIG. 2.4 – *Test 3. Temps d'apprentissage en fonction du nombre de centres.*

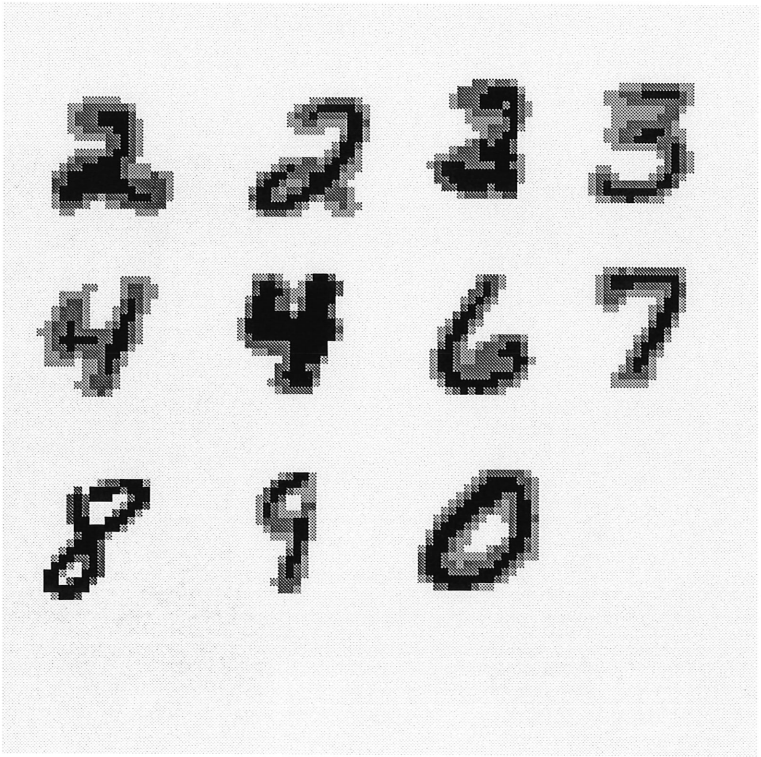

FIG. 2.5 – *Test 4. Exemples des centres finaux pris au hasard parmi tous les groupes. Chaque caractère représente l'image apprise par le neurone caché.*

Test 4. Taux de classification de la base d'apprentissage

Comme nous l'avons indiqué plus tôt, le taux maximum de classement correct sur la base d'apprentissage avec l'algorithme de base est de 89.36%. Cela représente jusqu'à quel point le réseau a appris les exemplaires de la base d'apprentissage. La figure 2.5 montre de façon visuelle ce que quelques centres pris aux hasard ont appris. On voit que le réseau est effectivement multimodal car quelques-uns des centres montrés sont des caractères de la même classe mais écrits différemment.

Test 5. Taux de classification de la base de tests

Comme indiqué précédemment, le taux maximal de réussite sur la base de test est de 81.57%. C'est environs 8% de moins que sur la base d'apprentissage. Cela signifie que lorsque le réseau examine des chiffres écrits d'une manière différente de ce qu'il a appris, environ 4 caractères sur 5 sont reconnus correctement.

Tous les chiffres ne sont pas aussi faciles à traiter par le réseau RBF. Certains sont reconnus à près de 100% dans le meilleur cas, d'autres ont un taux aussi bas que 16% même lorsque le taux moyen est de 58%. Le tableau 2.1 montre les taux moyens de reconnaissance pour chaque chiffre sur la base de test.

Ces résultats sont similaires à l'analyse faite dans [27] et en particulier dans [28] pour la différenciation de chaque classe.

Chiffre	% Reconnus
0	88.01
1	95.24
2	88.15
3	90.82
4	85.33
5	65.02
6	87.41
7	78.84
8	72.83
9	65.60
Total	81.57

TAB. 2.1 – *Taux de reconnaissance de chaque classe.*

Test 6: Apprentissage d'une base de données bruitée

Nous voulons vérifier la capacité du réseau à généraliser. Pour ce faire nous avons soumis la base d'apprentissage à un bruit normal et ensuite fait apprendre cette base au réseau. Celui-ci obtient un taux de classification correct de 87.12%, soit 2.24 % de moins que pour une classification normale, toute autre condition étant inchangée. (voir figure 2.6.) On voit que la qualité de la base d'apprentissage s'est dégradée mais que le réseau résiste assez bien au bruit.

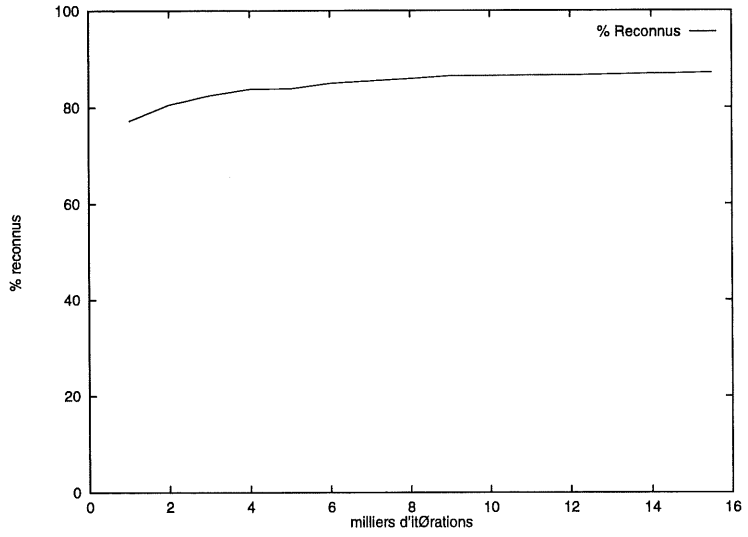

FIG. 2.6 – *Test 7. Taux de classification de la base d'apprentissage bruitée.*

Test 7. Classification de données bruitées.

Nous avons soumis la base de tests au même type de bruit que dans la section précédente. Le taux de classification correcte diminue ici aussi d'environs 2%

avec une base d'apprentissage bruitée ou non. Cela montre que le réseau se sert de sa capacité de généralisation pour filtrer le bruit.

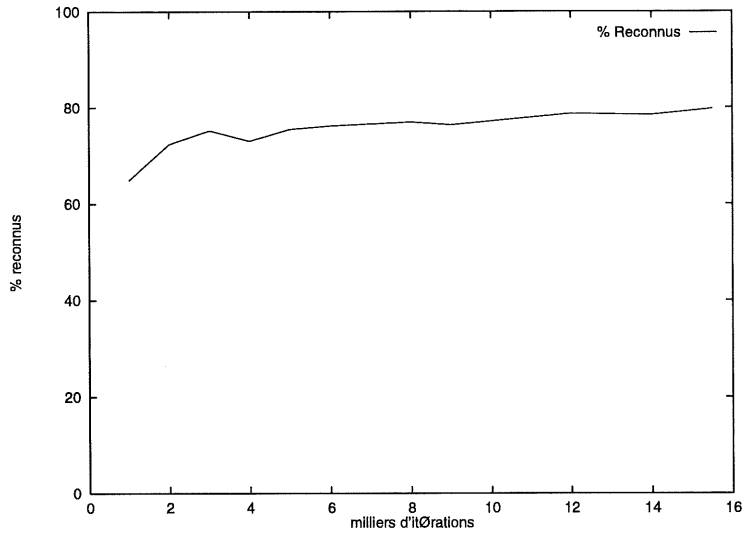

FIG. 2.7 – *Test 8. Taux de classification de la base de test bruitée.*

2.4.1 Performances

Pour un taux de reconnaissance raisonnable (89.36%), on a besoin de 323 secondes d'apprentissage au minimum soit environs 16000 itérations. La vitesse de rappel est linéairement proportionelle au nombre de neurones dans la couche cachée. Pour 120 neurones, le temps de rappel est de 13 secondes soit 0.005 secondes (moins d'un centième de seconde) par exemplaire.

Considérant que la base d'apprentissage contient assez peu de données et que celles-ci sont simplement un sous-échantillon du dessin original, la qualité du résultat est très bonne.

2.5 Comparaison au perceptron multicouche

On a dit dans l'introduction que le réseau RBF se compare avantageusement au perceptron multicouche. En effet, un perceptron à une couche cachée appliqué à nos données requiert que l'on ajuste 6685 poids et biais lors de l'apprentissage. Chaque ajustement des opérations complexes et est non-linéaire.

De son côté le réseau RBF requiert 31920 poids mais qui sont ajustés de façon linéaire. La vitesse du réseau RBF est supérieure lors de l'apprentissage mais légèrement inférieure à celle du perceptron lors du rappel car il faut effectuer un grand nombre de calcul de distances plutôt que de simples multiplications matricielles.

A titre de comparaison nous avons utilisé le simulateur de réseau SNNS dont le perceptron multicouche est considéré comme étant rapide [29]. Pour obtenir le même taux de classement sur la base d'apprentissage que le réseau RBF (soit 89.36 %) SNNS requiert 10 heures de travail. Il y a nécessairement du temps gaspillé par le simulateur car il s'agit d'un logiciel très complexe et aussi nous n'avons pas tenté d'obtenir une architecture optimale du perceptron donc le temps de travail peut diminuer considérablement, mais sera toujours d'un ordre de grandeur plus lent que le réseau RBF.

De plus la capacité de généralisation et de resistance au bruit du perceptron multicouche est moindre que celle du réseau RBF. En utilisant le simulateur SNNS sur nos données bruitées on obtient une perte du taux de classification correcte de l'ordre de 8%.

2.6 Comparaison à la classification bayesienne

Le réseau RBF et la classification bayésienne ne se comparent pas facilement directement dans la plupart des cas. Il aurait été intéressant d'utiliser un logiciel bien connu du milieu et bien distribué afin d'obtenir une comparaison concrète entre la méthode bayésienne et le réseau RBF. Cependant on a trouvé peu de logiciels de classification bayésienne dans le domaine public et de ceux-ci, aucun qui aurait été modifiable pour accepter nos données. Les raisons sont en partie le manque d'expertise avec la méthode bayésienne, ainsi que le temps qui nous a fait défaut pour s'assurer d'avoir un logiciel sans erreurs. Nous n'avons donc pas fait d'expériences pour comparer le réseau RBF et la classification bayésienne et donc nous n'avons pas de chiffres pour appuyer cette comparaison directement.

On peut cependant comparer nos résultats avec ceux de méthodes de classification bayésiennes utilisant la règle du plus proche voisin. Un exemple intéressant de cet algorithme est basé sur la théorie de Dempster-Shafer [18]. Cet algorithme imite le fonctionnement de l'algorithme des k plus proches voisins votants pour décider de la classe d'un exemplaire. En effet le classificateur D-S se sert des plus proches voisins comme éléments de preuve que l'exemplaire examiné fait partie d'une classe.

Supposons que l'on utilise ce modèle pour faire notre classification, avec notre base d'apprentissage de 1237 exemplaires de 256 caractéristiques.

Pour classer un exemplaire parmi 10 hypothèses possibles, cette méthode doit

procéder comme suit: Pour chaque hypothèse, on combine tous les éléments de preuve pour l'appartenance de l'exemplaire a cette classe. Cela signifie que pour chaque élément de cette classe dans la base d'apprentissage on ajoute une quantité de probabilité que notre exemplaire appartienne à cette classe selon la distance entre notre exemplaire et l'élément de la base d'apprentissage. Après normalisation, on obtiens les valeurs de croyance et de plausabilité pour chaque hypothèse.

Grâce à ces valeurs on peut faire la décision pour la classification en se basant soit sur la croyance ou sur la plausabilité et en utilisant les estimations de coût pour aider ce choix.

Tous ces calculs sont complexes. Avec notre base d'apprentissage il faudrait au moins 1237 calculs de distances, 1237 opérations de combinaison complexes et cela sans même prendre en compte la normalisation. Par comparaison l'algorithme RBF ne fait que 120 calculs de distance et 120 calculs de fonctions de réponse. C'est beaucoup moins que ce que requiert la méthode D-S. En réalité la quantité de calculs à faire dans D-S augmente de façon exponentielle en fonction du nombre d'éléments de la base d'apprentissage.

Evidemment on peut facilement réduire le temps de calcul en diminuant le nombre d'exemplaires dans la base d'apprentissage mais il faut pour cela l'épurer, ce qui peut diminuer la qualité du résultat. Il ne faut pas oublier que la méthode D-S doit conserver sa base d'apprentissage en mémoire en tout temps.

La méthode D-S a cependant certains avantages. Par exemple elle réagit bien

aux données mal étiquetées dans la base d'apprentissage. Elle semble aussi être plus précise que d'autres méthodes de classification par k-plus proches voisins tel que le k-plus proche voisins votants.

L'algorithme RBF est donc nettement plus économe en espace et en temps que la méthode D-S. Les études [31] et [30] montrent que pour les données du NIST, le résultat de RBF est légèrement moins précis que le résultat de variantes de la méthode bayésienne. Les deux méthodes offrent sensiblement la même résistance au bruit, mais en général RBF requiert moins de temps de calcul. Chaque méthode offre donc des avantages différents, mais le fait que RBF ne requiert pas de conserver toutes les données en mémoire est un avantage important.

Chapitre 3

Modifications du réseau

3.1 Les variantes du réseau RBF

Il y a plusieurs modifications possibles sur le réseau RBF de base. Par exemple, les modifications peuvent se situer au niveau de la couche cachée, la couche de sortie, ou dans la fonction de coût. L'utilisation d'une ou plusieurs de ces modifications pourrait améliorer les performances du réseau et le rendre plus général. Voici une liste incomplète des types de modifications possibles sur le réseau RBF:

- modifications sur le codage des entrées
- modifications sur le nombre de groupes

- modifications sur les fonctions de base
- modifications sur le choix du rayon
- modifications sur la couche de sortie
- modifications sur le codage des sorties
- modifications sur la fonction de coût
- modifications sur l'algorithme d'apprentissage
- modifications sur l'algorithme de regroupement
- modifications sur les connexions entre les couches
- etc.

Par exemple la fonction de coût peut être modifiée en ajoutant un facteur de lissage global ou local [14]. L'utilisation d'un facteur de lissage permet d'atténuer les effets du surapprentissage et de fonctions apprises trop raboteuses. La section 3.3.1 analyse si le lissage est utile pour notre application.

Nous nous intéressons surtout aux modifications sur la couche cachée car les modifications sur la couche de sortie ont relativement peu d'impact sur le résultat. Par contre, il est possible d'utiliser tout autre architecture pour la couche de sortie. Nous verrons à la section 3.1.2 ce qui nous motiverait à faire de la sorte. Pour l'instant, examinons certaines des modifications de la couche cachée plus en détail.

3.1.1 Les modifications sur la couche cachée

Une modification très répandue utilise des fonctions de base fixes, c'est à dire que la position des centres est égale à un des exemplaires de la base d'apprentissage et on n'ajuste pas cette position. Lorsque l'apprentissage le requiert, on utilise un exemplaire différent comme position du centre plutôt que de déplacer le centre dans l'espace. Autrement dit l'apprentissage ne fait que rechercher les prototypes les plus représentatifs pour fixer les centres. Les calculs sont considérablement simplifiés. En analysant la base d'apprentissage de caractères manuscrits nous avons observé que les classes sont facilement séparables, ainsi il peut être intéressant d'utiliser cette méthode. Malheureusement cette méthode ne peut pas être utilisée de façon dynamique satisfaisante et elle requiert beaucoup de mémoire puisqu'il faut conserver tous les exemplaires en mémoire.

Une méthode plus puissante, la sélection avant, démarre avec un ensemble vide et ajoute une fonction de base à la fois - celle qui réduit le plus la somme de l'erreur carrée - tant qu'un certain critère sur l'erreur cesse de décroître. La méthode de sélection avant n'est évidemment pas un algorithme linéaire mais elle offre les avantages suivants:

- il n'est pas nécessaire de fixer le nombre d'unités cachées à l'avance.
- le critère de sélection des groupes est explicite.
- les ressources de calcul nécessaires sont relativement faibles.

Cette méthode est très attrayante et nous avons retenu son principe pour améliorer le réseau. (voir section 3.1.1.)

Une autre méthode est l'élimination arrière qui fait le cheminement inverse, soit d'éliminer des groupes en minimisant l'erreur [19].

Une catégorie de modifications qui a une grande influence sur la performance du réseau est celle des modifications sur l'algorithme de regroupement. On peut par exemple choisir un nombre différent de voisins dans l'algorithme k-NN. La méthode de base utilise le plus proche voisin, c'est donc un algorithme 1-NN, mais on peut utiliser plus d'un voisin. En fait, la qualité de la classification augmente avec k jusqu'à une certaine limite assez basse. Il n'est donc pas nécessaire d'examiner de grands voisinages. Par contre, il faut trouver une méthode algorithmique pour trouver les plus proches voisins rapidement et une méthode pour décider quelle classe représente le mieux les k plus proches voisins [15].

Les modifications existent aussi directement sur l'algorithme k-moyennes et pas seulement sur ses paramètres. Il existe plusieurs de ces modifications, mais voici une variante de l'algorithme k-moyennes nommée k-moyennes par lots (Batch K-Means). Le k-moyennes par lots converge plus rapidement que le k-moyennes séquentiel et il nous donne une méthode pour déterminer quand arrêter le calcul [6]. Cet algorithme fonctionne comme suit:

Soit c_k le centre le plus près d'un exemplaire x_n. Pour chaque centre c_k on calcule la moyenne des $x_n \in X$ qui tombent dans ce centre pour tous les x_n.

On ajuste ensuite la position de c_k avec la moyenne calculée. Autrement dit:

$$\Delta c_k = \sum_{n=1}^{N} \frac{1}{N_k}(x_n - c_k)1_{c_k}(x_n)$$

où

$$1_{c_k}(x_n) = \begin{cases} 1 & \text{si } c_k = \text{plus proche centre de } x_n \\ 0 & \text{sinon.} \end{cases}$$

N est le nombre total d'exemplaires et $N_k = \sum_{n=1}^{N} 1_{c_k}(x_n)$ est le nombre d'exemplaires qui sont tombés dans le centre c_k.

L'algorithme k-moyennes par lots requiert environs la moitié du temps de calcul de l'algorithme séquentiel pour arriver au même résultat. En pratique k-moyenne séquentiel est plus rapide lors de la première époque mais k-moyennes par lot est beaucoup plus performant pour les époques suivantes.

L'expérimentation montre que l'algorithme atteint une solution satisfaisante après seulement 10 itérations.

L'algorithme k-moyennes locales avec état

Une façon simple d'améliorer la performance de l'algorithme de regroupement pour RBF est de donner à chaque groupe une variable d'état [16]. Cette variable d'état contient des statistiques sur le comportement du centre à laquelle elle est attachée. Plus précisément, nous pouvons suivre le nombre de fois (q_k)

où le centre c_k de la classe h a gagné un exemplaire sur les groupes concurrents.

Cette information d'état est très utile. Elle peut nous servir pour adapter le nombre de groupes à la densité locale des données. Ceci est accompli en mettant à jour la configuration des groupes après chaque époque (ou lorsqu'un grand nombre de données sont entrées), en éliminant les groupes qui gagnent très peu et divisant ceux qui ont gagnés trop souvent:

Pour chaque groupe c_k après chaque époque:

$$\text{si } q_k < N\theta_- \quad , \quad \text{éliminer le groupe.} \tag{3.1}$$

$$\text{si } q_k > N\theta_+ \quad , \quad \text{générer un nouveau groupe} \tag{3.2}$$

où N est le nombre d'exemplaires de la base d'apprentissage et $\theta_-, \theta_+ \in [0,1]$ sont deux barrières choisies de sorte que $(\theta_+ - \theta_-)N > 1$. Les limites θ_- et θ_+ sont choisies comme seuils qui indiquent si on doit respectivement éliminer ou ajouter un centre.

Si, après une époque l'algorithme indique qu'il faut changer le nombre de groupe, on remet toutes les valeurs q_k à zéro avant de commencer l'époque suivante. On continue cet algorithme jusqu'à ce que le nombre de centres soit stable sur plusieurs époques. Il faut évidemment n'éliminer qu'un centre à la fois sinon le modèle oscille et ne trouve jamais la bonne solution.

Un bon choix du nouveau centre est nécessaire pour assurer la convergence [16]. En pratique on peut simplement choisir un $x_i \in G_{c_k}$ comme position du nouveau centre. Une technique qui peut aider à générer les nouveaux centres

est d'utiliser une variante de l'algorithme de minimisation non-déterministe proposé par Girosi [20]. Cet algorithme permet de quitter un minimum local de l'erreur MSE en 'creusant un tunnel' sous les collines du gradient. Ainsi on peut facilement trouver une autre solution valide qui peut être meilleure sans avoir à tester plusieurs solutions intermédiaires. L'algorithme se déroule comme suit: Lorsque l'on cherche à poser un nouveau centre, on recherche parmi les exemplaires celui qui se trouve le plus près du cercle défini par le centre c_k et son rayon σ_k et non pas le plus près possible du centre c_k. Si l'erreur MSE diminue en posant un nouveau centre à cet endroit alors on garde ce centre. Sinon on recommence la recherche en doublant le rayon du cercle. La raison pour laquelle on cherche parmi les exemplaires est qu'ainsi on ne risque pas de poser le nouveau centre à un endroit où il n'y a pas de données.

Les valeurs θ nous donnent un contrôle sur le surapprentissage et l'uniformité de la distribution des centres en décidant quand il faut générer de nouveaux centres ou en éliminer. Les avantages d'utiliser cet algorithme se situent surtout au niveau de la sélection du nombre de centres de façon dynamique. Contrairement à une méthode par algorithme génétique, il n'est pas nécessaire de générer plusieurs solutions ou de garder les données en mémoire [8]. De plus, cet algorithme a une version par lots qui est très performante. C'est ce genre d'algorithme qui a été retenu pour cette étude car il dépasse de loin en performance un algorithme de style séquentiel.

Il est clair que cet algorithme converge lorsque $\theta_- = 0$; dans ce cas l'algorithme ajoute des groupes jusqu'à ce que chacun contienne moins que $N\theta_+$ points. Le

nombre de groupes augmente de façon monotone limitée par le nombre de points de données. En pratique il faut donner une petite valeur positive à θ_- pour éviter que les groupes soient attirés par des points aberrants.

Cet algorithme nous fournit plusieurs critères d'arrêt pour l'apprentissage. Il faut que le nombre de groupes soit stabilisé. Lorsque cet état est atteint, il faut que l'erreur de classification atteigne un minima local, ce qui devrait être assez rapide lorsque les groupes sont stables. Si les groupes sont instables, c'est que la solution n'est pas un minimum local.

Le problème de la distance

Les algorithmes de regroupement, tels que k plus proches voisins et LVQ ont tous en commun un problème qui peut être très sérieux. Supposons que les vecteurs d'entrées $X = \{x_1, x_2, x_3, x_4, x_5, x_6\} \subset R^2$ contiennent les classes $A = \{x_1, x_2, x_3\}$ et $B = \{x_4, x_5, x_6\}$ tel que montré dans la fig 3.1. La position initiale des centroïdes v_1 et v_2 est aussi montrée. Puisque le centre est plus près des quatre points restant que v_1, ces quatre points modifient tous v_2 seulement, v_1 ne changera pas lors de la première passe. De plus puisque les méthodes de mise à jour font glisser le centre v_2 d'une certaine distance vers la combinaison convexe des points, la chance que v_1 soit mis à jour est très faible. Même si cet algorithme donne une solution optimale locale, ce n'est pas une solution désirable.

Pour éviter ce problème on peut décider de modifier à la fois les groupes

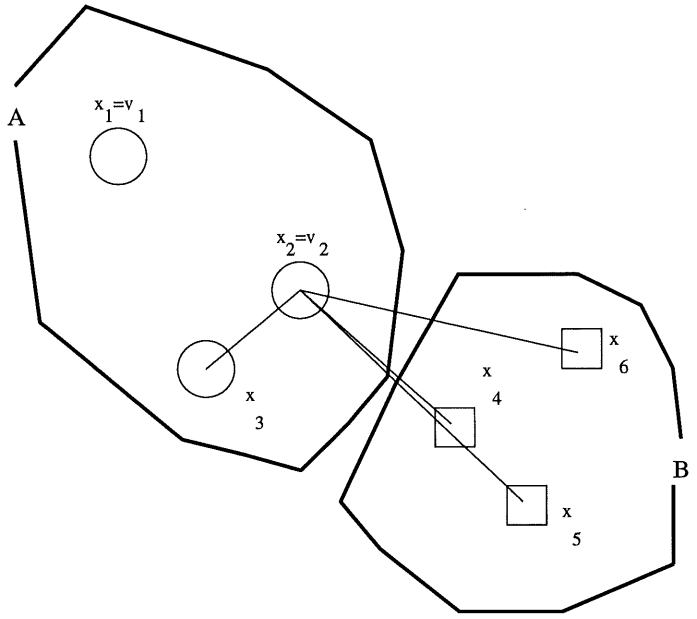

FIG. 3.1 – *Le problème de la distance, où v_1 est rarement mis à jour.*

gagnants et les groupes perdants en fonction de la distance entre x et les centres [21]. Les formules de mise à jour deviennent:

$$c_{k,t} = c_{k,t-1} + \eta_t(x - c_{k,t-1})\frac{D^2 - D + ||x - c_{k,t-1}||^2}{D^2}$$

pour le centre gagnant k où $D = \sum_{r=1}^{K} ||x - c_r||^2$ et

$$c_{j,t} = c_{j,t-1} + \eta_t(x - c_{j,t-1})\frac{||x - c_{i,t-1}||^2}{D^2}$$

pour tous les autres centres. Évidemment on évite de faire tout le calcul si $\frac{||x-c||^2}{D^2}$ est négligeable. Ces modifications aux formules de base proviennent de l'étude d'une fonction d'erreur par mauvaise classification locale. La première formule diminue l'erreur lorsque l'exemplaire est bien classé en approchant le centre de cet exemplaire. La seconde cherche à minimiser l'erreur en éloignant les centres qui ne sont pas gagnant de cet exemplaire en se servant de la fonction de coût pour pondérer ce déplacement.

Ainsi, même les centres perdants ont une chance de s'approcher des exemplaires, ce qui accélère la recherche d'une solution.

Calcul des distances

On a indiqué dans la section 2.4.1 que le calcul de la distance est une des opérations les plus coûteuses lors de l'apprentissage et du rappel. Une légère amélioration à ce niveau aura des répercutions importantes sur la performance du réseau.

Le calcul de la distance est une opération simple et on ne peut pas la rendre plus performante. On peut par contre utiliser une somme partielle pour certains calculs. Lorsque l'on veut trouver le centre c le plus près d'un x, si les n premiers termes de la somme

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 + \cdots + (x_n - c_n)^2$$

sont plus grands que la distance au carré $x - c$ minimale trouvée jusqu'à présent, alors le prototype c_j ne peut être plus près de x_i que ne l'est c_k et donc on arrête plutôt que de calculer les termes restant.

Nous avons toujours besoin de l'opérateur de calcul de distance complet, mais lorsque l'on recherche le centre c_k le plus près on diminue le nombre de calculs nécessaires.

3.1.2 Modifications sur la couche sortie

Il existe quelques variantes possibles pour la couche de sortie. Premièrement, il faut savoir qu'il n'est pas obligatoire d'utiliser un hétéro-associateur linéaire pour la couche de sortie. On peut se servir d'un autre algorithme de regroupement, ou d'un algorithme différent pour le perceptron, par exemple. Il n'est cependant pas certain que d'autres méthodes soient plus efficaces ou utiles que le perceptron.

On peut aussi faire des modifications au niveau de l'encodage du vecteur réponse. Par exemple, on peut utiliser une façon différente pour codifier le vecteur de sortie ou utiliser un seul neurone à sortie continue, entre autres. Cependant, pour le petit nombre de classes (10) nécessités pour ce modèle, la simple méthode un-de-x (un neurone est à 1 et les autres à 0) est la plus utile.

3.1.3 Autres variantes du modèle

Une méthode proposée par Billings et Zheng [8] pour trouver l'organisation optimale d'un réseau fait appel aux algorithmes génétiques. Ils se servent de l'ensemble des données d'apprentissage comme prototypes et l'algorithme cherche de façon parallèle quel sous-ensemble de ces exemplaires possède l'erreur MSE la plus petite. L'erreur d'apprentissage est plus petite mais l'erreur de généralisation plus grande que pour d'autres méthodes. Malgré le fait que cette méthode a l'avantage d'avoir un risque faible d'être coincée dans un minimum local de l'erreur, il faut cependant générer et tester une quantité de l'ordre

de $C_{210}^{1237} = 1237!/(210!1037!)$ réseaux. C'est une méthode qui demanderait un temps de calcul irréaliste et n'est pas dynamique.

La dernière variante possible sur la couche cachée est celle des réseaux RBF multicouches fonctionnelles. Ces réseaux utilisent plus d'une couche de neurones cachées. L'idée est de regrouper des centres en méta-groupes. Peu de littérature existe à ce sujet. Les avantages et inconvénients de cette variante sont donc inconnus.

3.2 Présentation du modèle amélioré

Pour le réseau amélioré, nous avons utilisé le k plus proches voisins par lots qui est la variante la plus utile. A partir de cet algorithme de base, nous avons ajouté une méthode de sélection avant, l'algorithme des k moyennes avec états, modifié pour fonctionner par lot. Nous avons finalement inclu la modification aux formules de déplacement des centres afin de régler certains problèmes causés par la distance. Nous sommes ainsi arrivés à un nouveau modèle qui possède beaucoup d'avantages provenant des modifications connues de RBF et peu de ses inconvénients. On nommera cet algorithme réseau à fonctions radiales de base munies d'états (FRBE).

Il y a eu peu d'études sur le comportement d'un tel algorithme en fonctionnement par lot. On sait que l'algorithme k plus proches voisins par lot fonctionne très bien mais aucune étude n'a prouvé mathématiquement jusqu'à présent que

1Tant que la solution n'est pas stable

2 Pour tous $x_i \in X$

3 c_k = plus proche voisin de x_i

4 s_i = classe de x_i

5 $D = \sum_{r=1}^{K} ||x - c_r||^2$

6 $\Delta c_k = \eta_t(x - c_k)\frac{D^2 - D + ||x-c_k||^2}{D^2}$

7 Pour tous $c_{l \neq k} \in C$

8 $\Delta c_l = \eta_t(x - c_l)\frac{|||x-c_l||^2}{D^2}$

9 $q_{k,s_i} = q_{k,s_i} + 1$

10 Pour tous les $c_k \in C$

11 s_k = classe majoritaire de c_k

12 s_l = classe secondaire de c_k

13 Si $q_{k,s_k} > \theta_{k+}$ divise c_k

14 Si $q_{k,s_l} > \theta_{k-}$ alors $\theta_{k+} = \theta_{k+} - 1$

15 Si $q_{k,s_k} < \theta_{k-}$ tue c_k

16 $c_k = c_k + (\frac{\Delta c_k}{N})$

TAB. 3.1 – *Algorithme d'agrégation de type épuration par lots.*

l'on puisse appliquer un algorithme de regroupement par lot qui garantit une réponse valable. Cependant les résultats que l'on a obtenus nous montrent qu'en pratique cela fonctionne très bien.

Notre algorithme est affiché dans le tableau 3.1 et est détaillé dans les sections suivantes.

3.2.1 Heuristiques pour le choix des θ

Si la méthode pour gérer le nombre d'éléments de chaque groupe est mal choisie, on risque d'avoir un algorithme où tous les groupes changent à chaque itération où qui ne trouve pas le nombre optimal de centres à utiliser. Il y a plusieurs façons d'utiliser les limites θ. On peut construire un algorithme utilisant une même paire θ_-, θ_+ pour tous les groupes, une paire pour chaque groupe, ou une paire pour chaque classe dans un groupe. La méthode choisie varie selon le problème à résoudre, la façon dont on veut contrôler le champ réceptif des neurones cachés et le nombre de groupes que l'on veut utiliser. Voici quelques possibilités plus détaillées:

Probabilités connues: Lorsque l'on sait d'avance quelle proportion de la base d'apprentissage appartient à chaque classe, on peut choisir les limites θ dans l'intention d'approcher le nombre d'exemplaires utilisés correctement du nombre d'exemplaires contenus dans cette classe. Par exemple si on a une classe s_h qui a une probabilité de 0.1, on peut choisir θ_+ équivalent a 10% du nombre d'exemplaires. Avec cette solution il n'y aurait qu'un groupe contenant tous les

exemplaires d'une classe, ce qui ne fonctionne pas en pratique. Une meilleure solution est d'utiliser un θ_+ différent pour chaque groupe, tous plus petit que l'espérance de leur classe associée. On définit dans ce cas l'espérance ainsi:

$$E = M - (O * \theta_-)$$

où M = nombre d'exemplaires dont la classe est s_h et O = nombre de centres dont la classe est s_h. On fait la somme des exemplaires bien classés pour tous les groupes d'une classe et on compare cette valeur avec l'espérance. S'il nous manque des exemplaires, on crée un nouveau centre avec une valeur θ_+ qui couvre la différence. On se sert de θ_- pour définir le moment ou la solution est assez près de l'espérance pour que l'on arrête le calcul. L'argument $(O * \theta_-)$ permet de minimiser les oscillations et d'éliminer des points excentriques. L'algorithme génère en moyenne 2 centres pour chaque groupe, ce qui améliore la capacité du réseau à faire de la séparation non-linéaire. Cependant cette méthode est assez complexe et requiert que l'on conserve de l'information à l'extérieur des neurones cachés. Elle demande aussi plus de calculs que les méthodes suivantes.

Minimisation des mauvais groupements: Cette méthode permet de *purifier* le plus possible les groupes pour que les éléments soient d'une seule classe. Le groupe appartient à la classe la plus représentative de ses éléments. Si une des autres classes est présente en nombre plus grand que θ_- on divise le groupe. On épure ainsi le contenu d'un groupe jusqu'à ce que les éléments des autres classes soient négligeables. Cette méthode est très simple et ne requiert pas de limite θ_+. Cependant on doit conserver un compteur pour toutes les classes dans chaque centre. De plus cette méthode ne maximise pas le champ réceptif et n'élimine pas les groupes qui sont trop petits, ce qui constituent

des inconvénients majeurs.

Total simple: La méthode la plus simple est de compter le nombre de fois que ce groupe a été choisi. Si ce nombre est plus grand que θ_+ alors il faut le diviser. Si ce groupe a gagné moins que θ_- fois alors il est à éliminer. Le problème principal de cette technique est qu'elle ne fonctionne que si la fonction à apprendre est très lisse. En effet, cette méthode amène les centres là où il y a une grande densité d'exemplaires, mais sans s'occuper de leur classe. Donc, si les exemplaires sont facilement séparable cela fonctionne très bien, sinon chaque groupe possèdera une grande quantité d'exemplaires de classes variées.

Épuration: Cette méthode maximise la taille du champ réceptif de chaque neurone tout en conservant l'aspect local. Elle minimise aussi le nombre de groupes et les ajuste à la densité locale des données. On utilise une paire θ_-, θ_+ par groupe. Lorsqu'un exemplaire fait partie du groupe, on augmente le compteur de classe du groupe pour cette classe.

On donne une petite valeur à θ_- et une grande valeur à θ_+. Le principe est qu'on part avec un grand champ réceptif pour un neurone caché et on le réduit progressivement jusqu'à ce que le groupe ne contienne que les exemplaires d'une classe. Cette manipulation ressemble un peu à celle de la 'minimisation des mauvais groupements' mais offre l'avantage d'ajuster le champ réceptif à son maximum possible. La manipulation se résume à ceci: Si la classe du groupe gagne plus que θ_+ on divise ce groupe. Si une classes qui n'est pas la classe principale du groupe est trop presente ($> \theta_-$) on réduit θ_+. Si la

classe du groupe gagne moins que θ_- alors on élimine ce groupe. On obtient ainsi de très gros groupes lorsqu'ils sont éloignés des autres classes et de petits groupes lorsque les données sont plus denses. On peut imaginer une méthode plus subtile lors de l'élimination que dans les autres méthodes. En effet, on peut par exemple examiner les exemplaires qui se sont libérés pour voir tout de suite s'ils vont faire dépasser la limiter θ_+ d'un autre groupe. Dans ce cas on peut décider d'augmenter le θ_+ de ce centre afin de diminuer le problème de l'oscillation décrit à la section 4.3.

Cette méthode est la plus puissante et ne requiert aucune valeur externe au neurone.

3.2.2 Calcul des σ

Nous avons vu dans la section 1.1 que les rayons de chaque centre sont calculés selon une heuristique au choix de l'utilisateur. On peut par exemple décider de prendre une fraction de la distance entre deux centres, choisir un rayon pour tous les centres ou un rayon par dimension par centre. Le choix final se retrouve souvent être simplement celui de l'utilisateur ou suggéré par expérimentation.

Le nouvel algorithme nous permet de faire un choix beaucoup plus éclairé de nos rayons. En effet, on peut choisir les rayons en fonction de la méthode retenue pour faire le regroupement. On peut prédire grâce aux valeurs θ quel sera le comportement du réseau pour certaines distributions de données et poser σ en conséquence.

En effet, σ dépend maintenant de l'algorithme qui a été sélectionné pour créer les groupes; par exemple si on a des groupes qui n'ont pas que des membres d'une seule classe (certains exemplaires sont dans des classes opposées) on peut décider de choisir un rayon plus petit, et donc le facteur de σ sera plus petit que 1.

On peut appliquer l'algorithme 3.2 pour fixer les rayons lorsque le regroupement ne crée que des groupes purs de points étrangers.

```
Pour chaque c1;
        trouver le centre le plus pres c2.
        trouver l'exemplaire de c1 le plus pres de c2
        trouver l'exemplaire de c2 le plus près de c1.
        calculer le point moyen entre ces 2 exemplaires
        rayon de c1 =distance au point moyen.
```

TAB. 3.2 – *Algorithme de choix du rayon lorsque les groupes possèdent très peu de points qui ne leur appartiennent pas.*

Puisque l'algorithme précis du choix de σ dépend de la méthode d'agrégation utilisée on ne peut pas tous les mentionner ici. On doit cependant remarquer que pour certaines méthodes telle que celle nommée *épuration*, que la méthode d'agrégation trouve d'elle-même le rayon optimal, car elle cherche à maximiser le champ réceptif jusqu'à ce qu'il rencontre une classe opposée. Dans ce cas, on peut donc utiliser un facteur σ très près de 1.

3.3 Avantages

Le FRBE a des avantages intéressants. Premièrement il converge vers une solution beaucoup plus rapidement que RBF avec la méthode de descente avec k plus proches voisins séquentiel. De plus on a un critère d'arrêt de l'algorithme lorsque l'erreur moyenne est minimale, contrairement au k plus proches voisins séquentiel où on utilise un nombre d'itérations fixe.

L'utilisation d'états nous fournit aussi deux nouveau paramètres pour contrôler la vitesse de convergence de cet algorithme. On peut utiliser ces paramètres pour revenir sur le calcul et voir si un autre taux peut trouver une meilleure solution.

Un avantage intéressant est que ce réseau est dynamisable, mais pas avec l'algorithme FRBE tel que présenté ici. En effet, si de nouveaux exemplaires nécessitent l'ajout d'un nouveau centre, il faut refaire l'apprentissage au niveau de la couche sortie. Mais on peut trouver des solutions intermédiaires très intéressantes du point de vue de la performance. Par exemple on peut utiliser l'algorithme FRBE pour faire l'apprentissage original et ensuite, puisque ce réseau contient les mêmes données qu'un réseau RBF, choisir un algorithme dynamique pour faire l'apprentissage en ligne.

Finalement, il est possible d'implanter cet algorithme sous forme de puce électronique. De ce point de vue, il serait moins pratique que la forme du RBF de base car il nécessite une mémoire pour faire l'apprentissage par lot, mais une fois que cette étape est terminée la seule limite devient le nombre de centres

possibles.

3.3.1 Propriété de lissage

Une des propriétés de cet algorithme est de pouvoir simuler le lissage. En effet, en utilisant la forme de l'algorithme de regroupement appropriée, on peut éliminer certains groupes et les regrouper en groupes plus importants. Cela a pour effet de diminuer la réponse du réseau à de petites modifications moins désirables. Par exemple, les points vraiment aberrants seront isolés dans l'espace et seront ignorés.

Cette forme de simulation du lissage est plus difficile à contrôler qu'une méthode plus traditionelle, mais elle a l'avantage de simplifier le réseau, ce qui offre un gain de performance. La méthode normale de faire du lissage est d'ajouter un terme d'atténuation à la fonction réponse du réseau. Cela ne simplifie pas le réseau et augmente la complexité des calculs.

Il faut noter que cette méthode n'est pas tout-à-fait équivalente au lissage traditionnel. En effet on peut utiliser les limites θ de façon à ignorer certaines parties de la fonction à apprendre en n'y posant pas de centres, tandis que dans le lissage traditionnel cette partie ne serait pas ignorée mais plutôt atténuée.

Chapitre 4

Analyse des résultats du nouveau réseau

4.1 Résultats obtenus

Nous avons fait le tests du chapitre 2 sur le réseau FRBE. Voici les résultats.

Expérimentations pour déterminer les valeurs des constantes

Comme pour le réseau de base, il y'a des paramètres à ajuster pour obtenir l'architecture optimale. Cependant ceux-ci ne sont pas tous les mêmes que dans le réseau de base.

Notons tout de suite les paramètres α et η qui ont les mêmes fonctions que dans l'algorithme RBF. L'expérience nous montre qu'ils prennent aussi sensiblement les mêmes valeurs que pour l'algorithme de base.

Nous avons ensuite expérimenté sur la valeur du facteur de σ, m_σ. Dans le chapitre précédent nous avons indiqué que l'algorithme trouve lui-même la valeur à donner à m_σ. Nous avons prouvé ce fait en multipliant σ par diverses valeurs m_σ dans l'intervalle $[0, 1]$ et en observant la qualité de la classification. La figure 4.1 nous montre ces résultats. Plus le m_σ s'approche de 1, meilleure est la classification. Cela indique donc que chaque groupe couvre tout l'espace disponible dans la plupart des cas. Ceci est une conséquence de la capacité qu'a le réseau d'adapter la densité des centres de diverses façon dans l'espace.

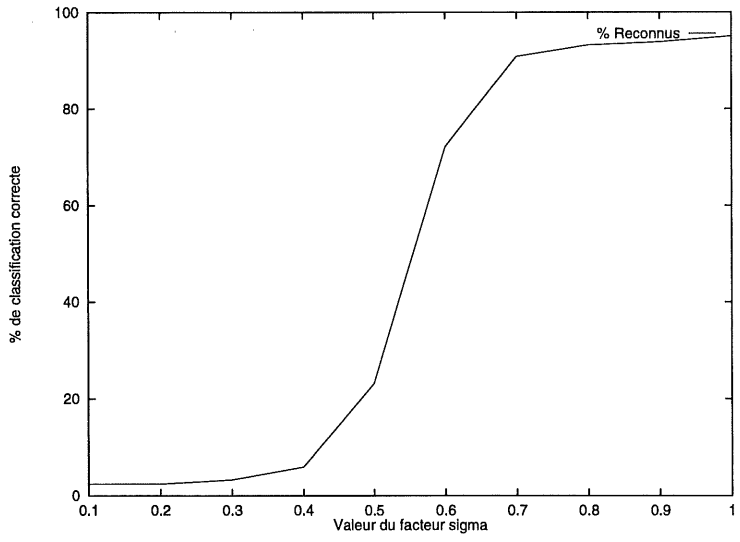

FIG. 4.1 – *Variation du taux de classification correcte en fonction de σ.*

Les autres paramètres sont les valeurs θ_- et θ_+. La valeur θ_+ ne sert que lors de la création de chaque nouveau centre. Tel qu'expliqué dans le chapitre

précédent, l'algorithme cherche à augmenter la taille de chaque groupe tant qu'il ne s'approprie pas des exemplaires d'autres classes. Donc la valeur de θ_+ n'as pas vraiment d'importance pour cette version de l'algorithme pourvu qu'il soit plus grand que θ_-. Nous avons utilisé $N\theta_+ = 14$ pour nos tests.

La valeur de θ_-, par contre, doit être fixée au départ. Les figures 4.2 et 4.3 nous montrent comment la qualité du résultat et le nombre de centres varient en fonction de θ_-. Ces résultats prouvent le fait que l'on peut ajuster la taille du réseau grâce au paramètre θ tel qu'énoncé dans la section 3.2.2. On peut donc ainsi obtenir un réseau plus petit et plus rapide en sacrifiant la qualité du résultat. Par exemple avec $N\theta_- = 9$ le taux de classement correct est 81.46% et la phase d'entraînement requiert 434 secondes tandis qu'avec $N\theta_- = 15$ le taux est de 72.557% pour un entraînement de 160 secondes (au même nombre d'itérations). Nous avons choisi de poser $N\theta_- = 2$ afin d'obtenir le meilleur résultat possible. Cette valeur nous permet de maximiser le nombre de centres tout en éliminant les points complètement isolés (voir la figure 4.2). Utiliser $N\theta_- = 0$ signifierait que l'on conserve tous les centres même lorsqu'on sait qu'ils ne représentent aucun exemplaire en entrée.

Test 2. Taux d'erreur de la classification sur la base d'apprentissage en fonction du nombre d'itérations

Comme pour le réseau RBF original, ce réseau obtient la meilleure classification à environs 15500 itérations avant de commencer à faire du surapprentissage. La raison est simple, c'est que ces itérations se font sur la couche de

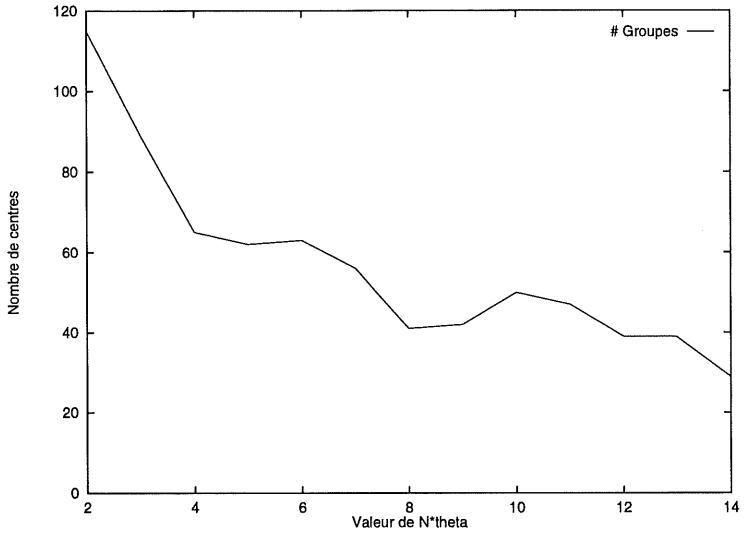

FIG. 4.2 – *Variation du nombre de centres en fonction de θ_-.*

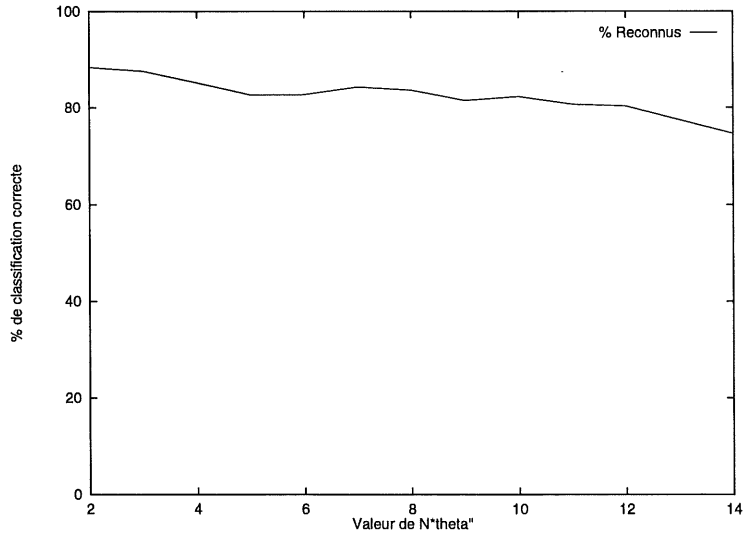

FIG. 4.3 – *Variation du taux de classification correcte en fonction de θ_-.*

sortie et elle n'as pas changée comparativement au réseau RBF.

Test 3. Taux d'erreur de la classification sur la base d'apprentissage en fonction du nombre de groupes

Puisque l'on ne peut choisir d'avance l'architecture du réseau FRBE, nous avons plutôt laissé le réseau détecter automatiquement le nombre adéquat de neurones cachés.

Le réseau avec états obtient un nombre de centres similaire à ce que nous avions obtenus par expérimentation de style essai-erreur pour le réseau original optimal, soit 115 en moyenne. Cela prouve que l'algorithme fonctionne correctement et est effectivement capable de trouver tout seul la meilleure architecture sans que l'usager ait à deviner le nombre de groupes à utiliser à priori.

Si on réexamine les figures 4.2 et 4.3, on voit que la qualité de la classification diminue en fonction du nombre de neurones cachés et que l'équilibre se trouve autour de 100 neurones cachés. La différence entre le nombre utilisé (120) et le nombre optimal (100) est faite de centres instables ou avec très peu de membres. On peut donc se limiter à 100 neurones sans perdre beaucoup d'exemplaires.

Test 4. Taux d'erreur sur la classification de la base d'apprentissage

On obtient un taux de reconnaissance de 96.36% comparé à 89.36% avec le réseau RBF pour le même nombre d'itérations (15500). C'est une différence de 7%, ce qui justifie amplement l'utilisation d'un apprentissage quatre fois plus lent.

Il faut signaler cependant qu'une dizaine des centres de FRBE, en moyenne, n'ont qu'un seul élément. Ceci est du à la méthode utilisée pour détecter la stabilité de la solution, à l'utilisation de $\theta_- = 2$ et aussi au fait que certains de ces exemplaires sont effectivement excentriques et donc il est normal qu'ils soient séparés des autres dans l'espace.

Test 5. Taux d'erreur sur la classification de la base de tests

Pour la classification de la base de tests, le résultat est de 89.78%, ce qui est supérieur de 8.41% à ce que l'on a obtenu pour le réseau RBF original. Le nouveau réseau a en effet une meilleure capacité à généraliser parce que ces centres sont mieux ajustés.

Test 6 et 7. Classification de données bruitées

Lors des tests d'apprentissage des données bruitées nous avons obtenus des résultats qui démontrent la valeur du nouvel algorithme. En effet, avec toutes les

conditions d'apprentissage égales le nouveau réseau obtient 95.88% de classification correcte sur la base d'apprentissage et 89.33% de classification correcte sur la base de test. Ces résultats sont 8.76% et 9.76% supérieurs aux résultats obtenu par le réseau RBF. C'est une amélioration très importante car par la pratique on a vu que l'on ne peut obtenir ce genre de résultat avec le réseau original.

Le réseau FRBE offre donc une meilleure capacité de résistance au bruit. La raison est que les centres ont un champ réceptif maximum tout en restant locaux et ont une meilleure position que dans l'algorithme de base. Le rayon de chaque centre étant le plus grand possible sans diminuer la qualité du résultat, une donnée bruitée a plus de chance de tomber à l'intérieur du rayon d'un centre et donc plus de chance de tomber à l'intérieur du bon groupe, donc le réseau permet à plus de données bruitées d'êtres classées correctement.

La solution fournie par le réseau avec la base d'apprentissage bruitée n'utilise que 96 groupes et 1362 secondes de calcul.

4.1.1 Performances

Le temps de calcul requis pour faire l'apprentissage est un désavantage comparativement au réseau de base. En effet le réseau RBF avec états requiert 1290 secondes de calcul pour faire l'apprentissage comparativement à 325 seconds pour le réseau original. C'est donc environs 4 fois plus. De plus il peut arriver que le nouveau réseau ne trouve pas de solution stable et donc qu'il ne termine

jamais. Ceci dépends des données du problème et se produisit environs 1 fois sur 10 avec nos données. Cela est une conséquence de la méthode pour détecter la stabilité ainsi que de la précision des calculs, entre autres, qui peuvent faire osciller le réseau entre deux ou plusieurs solutions valables.

Afin de justifier le temps de calcul plus long de l'algorithme FRBE, nous avons fait exécuter l'apprentissage du réseau de base quatre fois plus longtemps pour voir si les résultats s'amélioreraient avec plus de travail. Au contraire, le réseau original montre rapidement des signes de surrapprentissage (overtraining) qui diminue la qualité du résultat au déla de 16000 itérations, sans jamais obtenir de résultats aussi bon que ceux du nouvel algorithme.

4.2 Validité

Une des observations que l'on retire de ces expérimentations est qu'avec la base de données utilisée, les formules de calcul de distances de la section 3.1.1 s'avèrent presque inutiles. En effet, celles-ci sont plus utiles quand deux centres sont très près afin d'améliorer la convergence vers une solution. Puisque ces formules ajoutent de la complexité au calcul, il est préférable de ne les utiliser que lorsque nécessaire, si la fonction à approximer n'est pas très continue ou si il y'a peu d'exemplaires pour chaque centres.

Toutefois on voit que le réseau FRBE offre de très bonnes performances et un résultat très intéressant. L'augmentation de près de 10% de la classification

sur la base de tests bruitées justifie pleinement son utilisation et la capacité de pouvoir ajuster l'architecture du réseau selon le besoin est une caractéristique intéressante.

4.3 Problèmes

Malgré toutes les qualités du réseau FRBE il y a quand même plusieurs problèmes importants qui ne peuvent être ou ne sont pas résolus. Par exemple il est possible que l'algorithme, plutôt que d'améliorer la qualité du réseau à chaque époque, oscille entre deux ou trois états ou même plus. Cela vient du fait qu'il n'est pas garanti que l'algorithme simplifie la solution (réduit l'erreur ou l'énergie) à chaque étape. Il cherche plutôt un état où un sous-ensemble des exemplaires est couverts par des centres. Pour éliminer ce problème il nous faut un algorithme de regroupement qui évalue non seulement si tous les exemplaires sont bien couverts selon les critères fournis mais aussi que l'erreur MSE soit réduite à chaque fois que l'on fait un tel choix. Il nous faut donc évaluer si l'algorithme fait ce choix et sinon, en trouver un satisfaisant.

De plus il reste difficile de détecter le moment où le réseau a appris de façon satisfaisante, puisque les centres se déplacent et cela peut causer un ajout/retrait. Par exemple, on peut décider que le réseau est stable, ajuster la position des centres selon le k plus proches voisins par lots et se rendre compte que la solution n'est plus stable. Cela peut être dû à l'imprécision mathématique ou à un nombre d'exemplaires trop bas.

L'apprentissage de la couche de sortie se fait selon les exemplaires et non selon les centres. Cette couche simule un mode de fonctionnement un-de-x. Par contre chaque classe n'a pas la même probabilité et donc chaque neurone de sortie n'aura pas la même chance d'apprendre. Il est possible que cela diminue la précision de l'algorithme.

Pour obtenir de bons résultats avec RBF il faut s'assurer que les paramêtres de l'algorithme soient choisis en fonction de des données. Ce problème est encore présent dans FRBE mais à un degré moindre. Il reste des constantes dont l'évaluation ne se fait pas automatiquement et on ne sait pas exactement quoi choisir. Par exemple, θ, τ et le nombre d'itérations requis pour un apprentissage satisfaisant. Les limites θ nous permettent d'éviter en grande partie les problèmes de surapprentissage: on n'a qu'à ignorer tout groupe trop petit. Par contre un mauvais choix de θ peut causer un surapprentissage beaucoup plus grave que dans d'autres algorithmes. Il faut donc se méfier de cet effet.

Conclusion

Objectifs atteints

On a examiné le comportement de l'algorithme RBF lors de son utilisation pour la reconnaissance de caractères manuscrits. A partir de ces résultats on en a déduit un nouvel algorithme à l'aide de la littérature existante et de nos propres observations.

L'algorithme FRBE est plus rapide et demande moins de ressources que les autres méthodes avec une perte de précision minime. Il est multimodal et conserve son avantage sur les modèles probabilistes puisqu'il n'a pas besoin de conserver la base d'apprentissage en mémoire. Il est dynamisable. On peut donc effectivement envisager son utilisation à l'intérieur d'autres applications. L'étude de son comportement nous montre d'ailleurs qu'on peut faire un compromis entre la qualité du réseau et sa taille, et nos facteurs θ nous permettent justement de faire ce compromis selon les besoins.

L'algorithme d'agrégation permet de contourner en grande partie les problèmes de non-séparabilité. Lorsque les données sont distribuées de façon très dense dans l'espace, le bruit a plus de probabilité de causer des erreurs. L'algorithme de regroupement permet de diminuer ce taux moyen d'erreur en ajustant le nombre de groupes à la densité locale des données. La méthode empirique du choix des rayons est maintenant plus clairement liée à la distribution des données et on peut faire un choix mieux informé.

Objectifs futurs

Afin de régler ces problèmes et faire une étude plus approfondie du réseau FRBE, nous pouvons identifier les objectifs suivants:

Nous allons implanter ce réseau dans une application réelle pour examiner ses réactions. Il faudrait entre autres inclure les lettres de l'alphabet dans la base d'apprentissage et examiner comment le système réagit à un grand nombre de classes. On peut aussi s'intéresser à la reconnaissance de caractères accentués. Il faudrait de plus avoir un plus grand nombre d'utilisateurs afin d'avoir des données plus variées.

Nous pourrons examiner les autres variantes possibles pour voir lesquelles peuvent s'appliquer au nouveau réseau avec profit, en particulier des modifications sur la couche de sortie. On voudra aussi tester les algorithmes de regroupement. Entre autres, il faut tester l'algorithme sur un problème où les classes ne sont

pas connues d'avance mais où on peut faire de l'apprentissage supervisé (ex.: estimation d'une fonction). On se servira de l'algorithme d'agrégation ou de l'algorithme de regroupement.

Nous avons développé un algorithme auto-organisé qui arrive à une bonne solution et qui allège la plupart des problèmes des algorithmes de sa catégorie. Le comportement d'algorithmes par lots pour les réseaux de neurones artificiels tels que définis dans ce travail n'as pas encore été le sujet d'études mathématiques complètes. Bien que l'algorithme fonctionne en pratique, nous n'avons pas les preuves mathématiques qu'il converge vers une bonne solution. Il est donc important de démontrer la validité de ces algorithmes mathématiquement.

Finalement, un objectif moins mathématique serait de comparer cet algorithme à des neurones biologiques naturels pour voir s'il y a quelques-uns de ces mécanismes qui se retrouvent dans la nature. On sait déjà que certains neurones ont un comportement radial similaire à ceux du réseau RBF, mais peut-être que la nature a trouvé un moyen de recréer le comportement par lots de l'algorithme FRBE. Il serait instructif de le découvrir.

Bibliographie

[1] T. Pham Dinh, S. Wang et A. Yassine, *Training a Multi-layered Neural Network with a Trust-Region Based algorithm*, Math. Modelling and Numerical Analysis, vol. 24, No. 4, pp. 523-553, 1989.

[2] M. Amani, G.B. Bénié, S. Wang et D. Ziou, *Application of the Spectro-Spatial Classification of Remote Sensing Images to Agriculture*, symposium of the Canadian Remote Sensing Society, Vancouver, pp. 291-294, 1996.

[3] S. Wang, *A reconstruction Based Self-Organization Model for Feature Extraction*, Proceedings of the 1996 World Congress on Neural Networks, pp. 428-432, San Diego.

[4] F. La Rue, G.B. Bénié, S. Wang et J. Beaubien, *Application d'un algorithme à base de réseaux de neurones à la cartographie forestière*, Proceedings of the 26th Int. Symposium on Remote Sensing of Environment, pp. 279-282, 1996, Vancouver.

[5] John Moody & Christian J. Darken, *Fast Learning in Networks of Locally-Tuned Processing Units*, Neural Computation 1, pp 281-294, 1989.

[6] Bottou L. et Bengio Y., 1994, *Convergence Properties of the K-Means Algorithms.*

[7] Xu L., Krzyźak A. and Yuille A., 1994, *On Radial Basis Function Nets and Kernel Regression: Statistical Consistency, Convergence Rates, and Receptive Field Size*, Neural Networks, Vol. 7, No. 4, pp. 609-628.

[8] Billings S. A. and Zheng G. L., 1995, *Radial Basis Function Network Configuration Using Genetic Algorithms*, Neural Networks, Vol. 8, No. 6, pp. 877-890.

[9] Sekuler R. and Blake R, 1990, *Perception*, McGraw-Hill.

[10] D.S. Broomhead and D. Lowe, *Multivariable functional interpolation and adaptive networks*, Complex Syst., vol. 2, pp. 321-355, 1988.

[11] Light W.A.,*Some aspects of radial basis function approximation.*, Approx. Theory, Spline Functions Applic., No. 356, pp. 163-190, 1992.

[12] Powell M.J.D., *Radial basis functions in 1990.*, Adv. Numer. Anal., vol. 2, pp. 105-210, 1992.

[13] C.Y. Suen, M. Berthod et S. Mori, *Automatic Recognition of Handprinted Characters: The State of the Art*, IEEE Transactions and Pattern Analysis and Machine Intelligence, vol. 68, n 4, pp. 469-487, 1980.

[14] Mark J.L. Orr, *Local Smoothing of Radial Basis Function Networks*, Proc. of the International Symposium on Neural Networks, Taiwan, 1995.

[15] R. Rovati, R Ragazzoni, Zs. M. Kovàcs, R. Guerrieri, *Adaptive Voting Rules for k-Nearest Neighbors Classifiers*, Neural Computations, n. 7, pp. 594-605, 1995.

[16] Jose L. Marroquin and Frederico Girosi, *Some Extensions of the K-Means Algorithm for Image Segmentation and Pattern Classification*, Centre for Biological and Computational Learning paper no. 079, 1993.

[17] Batuhan Uluğ, Jun Zhao and Standley C. Ahalt, *Feature Based Classification of SAR Data using RBF Networks*, Department of Electrical Engineering, Ohio State University, 1993.

[18] Thierry Denøeux, *A k-nearest neighbor classification rule basesd on Dempster-Shafer Theory*, IEEE Transactions on Systems, Man and Cybernetics, 25 (05), 1995.

[19] Mark J. L. Orr,*Introduction to Radial Basis Function Networks*,Centre for Cognitive science, Endinburg, Avril 1996.

[20] Frederico Girosi, *Some Extensions of Radial Basis Functions and their Applications in Artificial Intelligence*, Computers Math. Applic., Vol. 24, No. 12, pp. 61-80,1992.

[21] Nikhil R. Pal, James C. Bezdek, Eric C.-K. Tsao, *Generalized Clustering Networks and Kohonen's Self-Organizing Scheme*, IEEE Transactions on Neural Networks, Vol. 4, No. 4, Juillet 1993.

[22] A. Belaid et Y. Belaid, *Reconnaissance des formes, méthodes et applications*, InterEditions, 1990.

[23] T. Cover, P.E. Hart, *Nearest neighbour pattern classification*, IEEE Transactions on Information Theory, Vol. 13, pp 21-27, 1967.

[24] J.C. Anigbogu and A. Belaid, *Recognition of multifont text using Markov models*, Proceedings of the 7th SCIA, Danemark, pp 13-16, 1991.

[25] F. Rosenblatt, *The Perceptron: a perceiving and recognizing automaton*, Project PARA, Cornell Aeronautic Laboratory Report, 85-460-1, 1957.

[26] C.K. Chow, *Statistical independence and threshold functions*, IEEE Transactions of Electrical Computer, Vol. 14. pp 66-68, 1965.

[27] R.G. Casey et H. Takahashi, *Experience in segmenting and recognizing the NIST database*, Proc. Int. Workshop Frontiers Handwriting Recognition, 1991.

[28] P.J. Grother, *Cross validation comparison of NIST OCR databases*, D.P.D'Amato, Vol. 1906, SPIE, 1993.

[29] Institute for Parallel and Distributed High Performance Systems, *Stuttgart Neural Network Simulator User Manual version 4.1*, rapport no. 6/95, Universit'e de Stuttgart, 1996.

[30] H. Yan, *Comparison of multilayer neural network and nearest neighbor classifiers for handwritten digit recognition*, International Journal of Neural Systems, Vol. 6, No. 4, Decembre 1995.

[31] C.L. Wilson, P.J. Grother et C.S. Barnes, *Binary decision clustering for neural-network-based optical character recognition*, Pattern Recognition, Vol. 29, No. 3, pp. 425-437, 1996.